UNIVERSITAT POLITÈCNICA DE CATALUNYA

DEPARTAMENT DE MATEMÀTICA APLICADA IV

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE DES SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

P H D T H E S I S

to obtain the title of

Doctor by the UPC

Programa de doctorat en
Matemàtica Aplicada

PhD of Science

of Université de Nice - Sophia Antipolis
Speciality: Computer Science

Defended by

Ignasi SAU VALLS

Optimization in Graphs under Degree Constraints Application to Telecommunication Networks

Grup de Grafs i Combinatòria
Advisor:
Xavier Muñoz López

Mascotte project, INRIA/CNRS-UNS
Advisors:
Jean-Claude Bermond, David Coudert

Version of July 22, 2009 – Will hopefully be defended on October 16, 2009

JURY:

Reviewers:	Pavol Hell	-	Simon Fraser Univ. (Vancouver, Canada)
	David Peleg	-	Weizmann Inst. of Science (Rehovot, Israel)
	Stéphan Thomassé	-	Univ. de Montpellier (Montpellier, France)
President :	Bruce Reed	-	McGill University (Montreal, Canada)
Examinators:	Jean-Claude Bermond	-	CNRS (Sophia-Antipolis, France)
	David Coudert	-	INRIA (Sophia-Antipolis, France)
	Xavier Muñoz	-	UPC (Barcelona, Catalonia, Spain)
	Miquel-Àngel Fiol	-	UPC (Barcelona, Catalonia, Spain)
Invited:	Stéphane Pérennes	-	CNRS (Sophia-Antipolis, France)

Optimization in Graphs under Degree Constraints. Application to Telecommunication Networks

Résumé: La première partie de cette thèse s'intéresse au groupage de trafic dans les réseaux de télécommunications. La notion de groupage de trafic correspond à l'agrégation de flux de faible débit dans des conduits de plus gros débit. Cependant, à chaque insertion ou extraction de trafic sur une longueur d'onde il faut placer dans le noeud du réseau un multiplexeur à insertion/extraction (ADM). De plus il faut un ADM pour chaque longueur d'onde utilisée dans le noeud, ce qui représente un coût d'équipements important. Les objectifs du groupage de trafic sont d'une part le partage efficace de la bande passante et d'autre part la réduction du coût des équipements de routage. Nous présentons des résultats d'inapproximabilité, des algorithmes d'approximation, un nouveau modèle qui permet au réseau de pouvoir router n'importe quel graphe de requêtes de degré borné, ainsi que des solutions optimales pour deux scénarios avec trafic all-to-all: l'anneau bidirectionnel et l'anneau unidirectionnel avec un facteur de groupage qui change de manière dynamique.

La deuxième partie de la thèse s'interesse aux problèmes consistant à trouver des sous-graphes avec contraintes sur le degré. Cette classe de problèmes est plus générale que le groupage de trafic, qui est un cas particulier. Il s'agit de trouver des sous-graphes d'un graphe donné avec contraintes sur le degré, tout en optimisant un paramètre du graphe (très souvent, le nombre de sommets ou d'arêtes). Nous présentons des algorithmes d'approximation, des resultats d'inapproximabilité, des études sur la complexité paramétrique, des algorithmes exacts pour les graphes planaires, ainsi qu'une méthodologie générale qui permet de résoudre efficacement cette classe de problèmes (et de manière plus générale, la classe de problèmes tels qu'une solution peut être codé avec une partition d'un sous-ensemble des sommets) pour les graphes plongés dans une surface.

Finalement, plurieurs annexes présentent des résultats sur des problèmes connexes.

Mots-clés: graphes, groupage de trafic, réseaux optiques, décomposition de graphes, optimisation, complexité, algorithmes d'approximation, complexité paramétrique, branch-width, programmation dynamique, graphes dans les surfaces.

Acknowledgements

Moltes gràcies als meus pares, a la Mar, als amics de (i per a) tota la vida i als membres del MA4 i de la UPC. I com no... gràcies Barça per haver-me donat tantes alegries!

Un grand merci à toute l'équipe MASCOTTE, à mes camarades niçois, à l'ensemble de mes colocs (immense mais fini quand même) et aux superbes joueurs de futsal du CSPF.

Muchas gracias a todos los que habéis contribuido incansablemente durante los últimos 4 años a que vaya a guardar para siempre un inmejorable recuerdo de mi etapa en Nice.

Thanks a lot to all my coauthors and scientific colleagues. This thesis is also theirs. A special mention goes to my three advisors for their trust in me and their generosity.

Muito obrigado também aos meus amigos brasileiros por os seus sinceros sorrisos.

Sophia-Antipolis
July 22, 2009

Contents

Overview of this Thesis

This thesis contains the results obtained during the last three years in both MASCOTTE project of INRIA/CNRS-UNS in Sophia-Antipolis (France) and Departament de Matemàtica Aplicada 4 of Universitat Politècnica de Catalunya in Barcelona (Catalonia, Spain). It is also the result of numerous research visits, like the ones performed at Deparment of Theoretical Computer Science of IMFM (Ljubljana, Slovenia), the Computer Science Department of Technion (Haifa, Israel), Departamento de Computaçao of UFC (Fortaleza, Brazil), the Deparment of Mathematics of NKU (Athens, Greece), and Algorithms Research Group of University of Bergen (Bergen, Norway). These results would not have been obtained without the invaluable collaboration of my coauthors, to whom I am deeply indebted.

This thesis is organized into two main parts plus four appendices containing some further contributions. We first provide in Part I some basic preliminaries and fix the notation to be used throughout the thesis. The concepts that are used only locally are defined in the corresponding chapters. Each of the two main parts (containing four chapters each, numbered from 1 to 8) begins with an introduction to the topic, an overview of the results in the literature, and a summary of our contributions (see Parts II and III). Finally, Part IV concludes the thesis and suggests some lines for further research.

The main topic of the thesis consists of graph optimization problems with constraints on the degree. Generally, these problems take as input a (weighted or unweighted) graph G and ask for a subgraph (or a set of subgraphs) of G satisfying certain degree constraints, while optimizing some parameter, usually the number of vertices or edges of the subgraphs. Most problems considered here are NP-hard, i.e., they are unlikely to be solvable by efficient exact algorithms. Therefore, it is important to design fast algorithms providing a solution which is provably close to the optimal solution. Nevertheless, sometimes it is possible to prove hardness results showing that certain algorithms cannot exist. Finally, it is also interesting to provide optimal solutions for some restricted classes of input graphs, and also to design *fast* (of course, not expected to run in polynomial time) exact or parameterized algorithms for a general input. The first part is devoted to traffic grooming.

Traffic grooming. Traffic grooming is a central problem in optical networks. Loosely speaking, it refers to packing low-rate signals into higher-speed streams, in order to improve bandwidth utilization and reduce network cost. The objective is to minimize the number of Add-Drop Multiplexers (ADMs), which are devices that insert/extract low-rate traffic to/from a high-speed stream. In graph-theoretical terms, the problem can be translated

into finding a partition of the edges of a request graph into subgraphs with bounded number of edges, the objective being to minimize the total number of vertices of the partition. The most used topology in real networks, like SONET WDM optical networks (see page 23), is the unidirectional or bidirectional ring.

We first focus in Chapter 1 on polynomial-time approximation algorithms and hardness results for a general request graph in the ring and path topologies. On the one hand, we provide the first inapproximability result for Ring and Path Traffic Grooming for fixed values of the grooming factor C (see page 23 for the definition), answering affirmatively to a conjecture in the literature. On the other hand, we provide a polynomial-time approximation algorithm for Ring and Path Traffic Grooming, based on a greedy cover algorithm, with an approximation ratio independent of C. This is the first approximation algorithm with this property, which is useful in practical applications since in backbone networks the grooming factor can be greater than the network size.

We introduce in Chapter 2 a new model of traffic grooming in unidirectional rings, in order to design networks being able to support *any* request graph with a fixed bounded degree. The existing theoretical models in the literature are much more rigid, and do not allow such adaptability. We show that the problem is essentially equivalent to finding the least integer $M(C, \Delta)$ such that the edges of any graph with maximum degree at most Δ can be partitioned into subgraphs with at most C edges and each vertex appears in at most $M(C, \Delta)$ subgraphs. We establish the value of $M(C, \Delta)$ for almost all values of C and Δ.

In Chapter 3 we focus on traffic grooming in bidirectional rings considering symmetric shortest path routing and all-to-all unitary requests, which had not been studied before. We formally state the problem, provide general lower bounds, and construct infinite families of optimal solutions for $C \in \{1, 2, 3\}$ and C of the form $k(k+1)/2$.

In Chapter 4 we study traffic grooming for two-period optical networks, a variation of the traffic grooming problem for WDM unidirectional ring networks that allows some dynamism on the traffic. In the two-period grooming problem, during the first period of time, there is an all-to-all uniform traffic among n nodes, each request using $1/C$ of the bandwidth; and during the second period, there is all-to-all uniform traffic only among a subset V of v nodes, each request now being allowed to use $1/C'$ of the bandwidth, where $C' < C$. Using tools of graph decompositions, we determine the minimum number of ADMs for any n, v and $C = 4$ and $C' \in \{1, 2, 3\}$.

As discussed above, the traffic grooming problem consists essentially in partitioning the edges of a graph into subgraphs with bounded number of edges, while minimizing the total number of vertices of the partition. In other words, the objective is to partition the edges of a graph into subgraphs maximizing the average *density* (defined as the edges-to-vertices ratio), or equivalently the average degree. The study of the traffic grooming problem leads naturally to the study of a family of graph-theoretic problems dealing with general constraints on the degree, such as the minimum degree or the maximum degree of the subgraphs. This is the topic of the second part of this thesis.

Degree-constrained subgraphs. For a typical degree-constrained subgraph problem, the objective is to find an optimal subgraph (usually, a subgraph with the maximum or

minimum number of vertices or edges) satisfying certain degree constraints, like bounded maximum or minimum degree.

We begin in Chapter 5 by studying the (classical) complexity of several families of degree-constrained problems, giving a variety of hardness results and polynomial-time approximation algorithms. Among others, we use the error amplification technique, probabilistic algorithms, and structural results of graph minors.

We then study in Chapter 6 the parameterized complexity of finding small degree-constrained subgraphs, when the parameter is the number of vertices of the subgraphs. We prove $W[1]$-hardness results in general graphs and provide explicit FPT algorithms (see page 18 for the definition of these terms) for H-minor-free graphs, using structural results and dynamic programming techniques.

Devising subexponential parameterized algorithms for degree-constrained subgraph problems on planar graphs is the topic of Chapter 7, which uses bidimensionality theory combined with novel dynamic programming techniques. As a result, we obtain subexponential parameterized and exact algorithms for several families of problems on planar graphs.

Finally, we provide in Chapter 8 a framework for the design of $2^{O(k)} \cdot n$ step dynamic programming algorithms for surface-embedded graphs on n vertices of branchwidth at most k. That way, we considerably extend the class of problems that can be solved by algorithms whose running times have a *single exponential dependence* on branchwidth, and improve the running time of several existing algorithms. Our approach is based on a new type of branch decomposition called *surface cut decomposition*, which generalizes sphere cut decompositions for planar graphs, and where dynamic programming should be applied for each particular problem. The existence of such algorithms is proved by a detailed analysis of how non-crossing partitions are arranged on surfaces with boundary and uses diverse techniques from topological graph theory and analytic combinatorics.

Further Contributions. The appendices of this thesis contain several articles whose motivation mostly originated from the problems discussed so far.

Appendix A deals with permutation routing and (ℓ, k)-routing on plane grids. The packet routing problem plays an essential role in communication networks. It involves how to transfer data from some origins to some destinations within a reasonable amount of time. In the (ℓ, k)-routing problem, each node can send at most ℓ packets and receive at most k packets. In other words, the request graph can be represented by a bipartite graph where the two independent sets have degree bounded by ℓ and k, respectively. Permutation routing is the particular case $\ell = k = 1$. In Appendix A.1 we provide an optimal distributed permutation routing algorithm on full-duplex triangular grids, and in Appendix A.2 we provide tight permutation routing and (k, k)-routing algorithms on plane grids, as well as approximation algorithms for the general (ℓ, k)-routing problem.

Appendix B is concerned with the problem of routing a set of requests in AOLS networks with the aim of minimizing the number of labels required to ensure the forwarding. This problem is in a sense similar to traffic grooming. In Appendix B.1 we study particularly this network design problem when the network is a path, providing an exact polynomial-time algorithm for the case in which all the requests have a common source

and some approximation algorithms and heuristics for an arbitrary number of sources. In Appendix B.2 we formalize the considered problem by associating to each routing strategy a logical hypergraph whose hyperedges are dipaths of the physical graph, that correspond to tunnels in GMPLS terminology. Such a hypergraph is called a *hypergraph layout*, to which we assign a cost function given by its physical length plus the total number of hops traveled by the traffic. Minimizing the cost of the design of an AOLS network can then be expressed as finding a minimum cost hypergraph layout. We prove the first hardness results in this area and propose approximation algorithms for the problem, inspiring ourselves from techniques that had been previously applied to VPL problems for ATM networks.

Tolerance graphs are the topic of Appendix C. Tolerance graphs model interval relations in such a way that intervals can tolerate a certain degree of overlap without being in conflict. This subclass of perfect graphs has been extensively studied, due to both its interesting structure and its numerous applications. In Appendix C.1 we propose the first non-trivial intersection model for general tolerance graphs, given by three-dimensional parallelepipeds, which extends the widely known intersection model of parallelograms in the plane that characterizes the class of bounded tolerance graphs. This new representation also enables us to improve the time complexity of algorithms for computing a minimum coloring, a maximum clique, and a maximum weight independent set. The recognition of tolerance graphs – namely, the problem of deciding whether a given graph is a tolerance graph – as well as the recognition of their main subclass of bounded tolerance graphs, have been the most fundamental open problems on this class of graphs since their introduction in 1982. In Appendix C.2 we prove that both recognition problems are NP-complete, even in the case where the input graph is a trapezoid graph. The presented results are surprising because, on the one hand, most subclasses of perfect graphs admit polynomial recognition algorithms and, on the other hand, bounded tolerance graphs were believed to be efficiently recognizable as they are a natural special case of trapezoid graphs, which can be recognized in polynomial time.

The existence of a circuit through a prescribed set of vertices is an important graph-theoretical question. In Appendix D.1 we study the following problem: which is the largest integer k such that, given any subset of k ordered vertices of an infinite square grid, there exists a circuit visiting the k vertices in the prescribed order using each edge at most once? We prove that $k = 10$. To this end, we first provide a counterexample implying that $k < 11$. To show that $k \geq 10$, we introduce a methodology, based on the notion of *core graph*, to reduce drastically the number of possible vertex configurations, and then we test each one of the resulting configurations with an ILP solver.

A graph parameter is *self-dual* in some class of graphs embeddable in some surface if its value does not change in the dual graph more than a constant factor. Self-duality has been examined for several width-parameters, such as branchwidth, pathwidth, and treewidth (see page 15). In Appendix D.2 we give a direct proof of the self-duality of branchwidth (denoted $\mathbf{bw}$) in graphs embedded in some surface. In this direction, we prove that $\mathbf{bw}(G^*) \leq 6 \cdot \mathbf{bw}(G) + 2g - 4$ for any graph G embedded in a surface of Euler genus g (see page 179 for the definition).

Finally, Appendix D.3 deals with a coloring problem in triangular grids. Namely, we are

given an induced subgraph G of a triangular grid together with a integer demand function on its vertices. The objective is to assign to each vertex as many colors as its demand, in such a way that adjacent vertices get disjoint sets of colors, while minimizing the total number of used colors. This minimum is called the *multichromatic number* of G. Finding the multichromatic number of induced subgraphs of the triangular grid has important applications in cellular networks, and has been widely studied during the last years. We provide a simple algorithm to color any triangle-free induced subgraph of the triangular grid with at most seven colors when each vertex has demand three. Our result simplifies and improves some existing results in the literature.

The bibliography distinguishes between the personal publications of the author and other articles (tagged as "General Bibliography"). The personal bibliography is classified into "International Journals" (refs. [J1-J3]), "Book Chapters" (refs. [B4-B5]), "International Conferences" (refs. [C6-C18]), "National Conferences" (ref. [N19]), "Submitted for Publication" (refs. [S20-S26]), and "In Preparation" (refs. [P27-P29]). Table 1 summarizes the coauthors associated to each chapter and appendix of this thesis, as well as the corresponding publications.

Chapter/Appendix	**Result of joint work with**	**Publications**
Chapter 1	O. Amini and S. Pérennes	[J2, C7, N19, B4]
Chapter 2	X. Muñoz and Z. Li	[C15, C13, P29]
Chapter 3	J.-C. Bermond and X. Muñoz	[C11, P28]
Chapter 4	J.-C. Bermond, C. J. Colbourn, L. Gionfriddo, and G. Quattrocchi	[S21]
Chapter 5	O. Amini, D. Peleg, S. Pérennes, and S. Saurabh	[C6, S20]
Chapter 6	O. Amini and S. Saurabh	[C8, P27]
Chapter 7	D. M. Thilikos	[C17, S25]
Chapter 8	J. Rué and D. M. Thilikos	[S24]
Appendix A.1	J. Žerovnik	[J3, C18]
Appendix A.2	O. Amini, F. Huc, and J. Žerovnik	[J1, B5]
Appendix B.1	J.-C. Bermond, D. Coudert, J. Moulierac, S. Pérennes, H. Rivano, and F. Solano	[C9]
Appendix B.2	J.-C. Bermond, D. Coudert, J. Moulierac, S. Pérennes, and F. Solano	[C10]
Appendix C.1	G. B. Mertzios and S. Zaks	[C14, S22]
Appendix C.2	G. B. Mertzios and S. Zaks	[S23]
Appendix D.1	D. Coudert and F. Giroire	[C12]
Appendix D.2	D. M. Thilikos	[C16]
Appendix D.3	P. Šparl and J. Žerovnik	[S26]

Table 1: Coauthors and publications associated to each part of this thesis.

Part I

Preliminaries

We provide here some basic preliminaries and fix the notation to be used throughout this thesis. This part is intended to be looked up when necessary, rather than to be read sequentially.

I.1 Graphs

We use standard graph-theoretical terminology, and we assume that the reader is familiar with the basic concepts of graph theory. For more details, see for instance the classical monograph of Berge [42] or the more recent book of Diestel [95].

Given a simple undirected graph $G = (V, E)$, and edge between the vertices u and v is denoted $\{u, v\}$, and then u and v are said to be *adjacent*. An edge is *incident* to its two endpoints. The *degree* of a vertex v in G is the number of vertices incident to v in G. Namely, $d(v) = |\{u \in V(G) : \{u, v\} \in E(G)\}|$. The *maximum degree* (resp. *minimum degree*) of a graph G is the maximum (resp. minimum) degree over all its vertices, and it is denoted $\Delta(G)$ (resp. $\delta(G)$).

For a graph $G = (V, E)$ and a subset $V' \subseteq V$, we denote the *induced subgraph* on V' by $G[V'] = (V', E')$, where $E' = \{\{u, v\} \in E : u, v \in V'\}$. For $v \in V$, we denote by $N_G(v)$ the *neighborhood* of v, namely $N_G(v) = \{u \in V : \{u, v\} \in E\}$. The *closed neighborhood* $N_G[v]$ of v is $N_G(v) \cup \{v\}$. In the same way we define $N_G[S]$ for $S \subseteq V$ as $N_G[S] = \cup_{v \in S} N_G[v]$, and $N(S) = N[S] \setminus S$. We may omit the subscript G if the graph is clear from the context.

A graph on n vertices is called *complete* if it contains an edge between each pair of vertices, and is denoted K_n. The complete graph on three vertices is known as the *triangle*. The *path* on n vertices $v_0, \ldots, v_{n-1}$ with the $n-1$ edges $\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{n-2}, v_{n-1}\}$ is denoted P_n. The *cycle* on n vertices obtained from P_n by adding the edge $\{v_{n-1}, v_0\}$ is denoted C_n. A graph G is k-partite if $V(G)$ can be partitioned into k classes $V_0, \ldots, V_{k-1}$ such that there are only edges between classes V_i and V_j with $i \neq j$. The 2-partite (resp. 3-partite) graphs are known as *bipartite* (resp. *tripartite*). The *density* ρ of a graph $G = (V, E)$ is its edges-to-vertices ratio, that is $\rho(G) = \frac{|E(G)|}{|V(G)|}$. More generally, for any subset $S \subseteq V$, we denote its *density* by $\rho_G(S)$ or simply $\rho(S)$, and define it to be the density of the induced graph on S, i.e., $\rho(S) = \rho(G[S])$.

I.1.1 Tree-like decompositions of graphs

We define some well-known decompositions of graphs that, loosely speaking, quantify the resemblance of a graph to a tree.

Tree decompositions. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T, \mathcal{X})$, where $T = (I, F)$ is a tree, and $\mathcal{X} = \{X_i\},\ i \in I$ is a family of subsets of $V(G)$, called *bags* and indexed by the nodes of T, such that

1. each vertex $v \in V$ appears in at least one bag, i.e., $\bigcup_{i \in I} X_i = V$;
2. for each $v \in V$ the set of nodes indexed by $\{i \mid i \in I,\ v \in X_i\}$ forms a subtree of T;

3. For each edge $e = \{x, y\} \in E$, there is an $i \in I$ such that $x, y \in X_i$.

The *width* of a tree decomposition, denoted by $w((T, \mathcal{X}))$, is defined as $\max_{i \in I}\{|X_i| - 1\}$. The *treewidth* of G, denoted by $\mathbf{tw}(G)$, is the minimum width of a tree decomposition of G. We refer to the survey of Bodlaender [58] for an introductory overview on treewidth and its use in algorithmic graph theory.

Path decompositions. A tree decomposition $(T, \mathcal{X})$ in which T is a path is called a *path decomposition*. The width of a path decomposition $(T, \mathcal{X})$ is the width of $(T, \mathcal{X})$ as a tree decomposition. The *pathwidth* is defined exactly as above by restricting to the path decompositions, and it is denoted $\mathbf{pw}$. A path decomposition $(T, \mathcal{X})$, with T a path of length n, is usually denoted by $(X_0, X_1, \ldots, X_n)$.

Branch decompositions. Let G be a graph on n vertices. A *branch decomposition* (T, μ) of a graph G consists of an unrooted ternary tree T (i.e., all internal vertices are of degree three) and a bijection $\mu : L \to E(G)$ from the set L of leaves of T to the edge set of G. We define for every edge e of T the *middle set* $\mathbf{mid}(e) \subseteq V(G)$ as follows. Let T_1 and T_2 be the two connected components of $T \setminus \{e\}$. Then let G_i be the graph induced by the edge set $\{\mu(f) : f \in L \cap V(T_i)\}$ for $i \in \{1, 2\}$. The *middle set* is the intersection of the vertex sets of G_1 and G_2, i.e., $\mathbf{mid}(e) := V(G_1) \cap V(G_2)$. The *width* of (T, μ) is the maximum order of the middle sets over all edges of T, i.e., $\mathbf{w}(T, \mu) := \max\{|\mathbf{mid}(e)| \colon e \in T\}$. An optimal branch decomposition of G is defined by a tree T and a bijection μ which give the minimum width, the *branchwidth*, denoted by $\mathbf{bw}(G)$.

Robertson and Seymour [173] proved that the branchwidth and the treewidth of a graph G, with $|E(G)| \geq 3$, satisfy $\mathbf{bw}(G) \leq \mathbf{tw}(G) + 1 \leq \frac{3}{2}\mathbf{bw}(G)$. Therefore, a family of graphs has bounded branchwidth if and only if it has bounded treewidth.

I.1.2 Graph minors

Let $G = (V, E)$ be a simple undirected graph and let $e = \{x, y\} \in E$. We define $E_G(v) = \{\{v, u\} \mid u \in N_G(v)\}$. We denote by $G \backslash e$ the graph G' where $G' = (V, E - \{e\})$ and we say that G' *occurs from* G *after an edge removal*. We also denote by G/e the graph G' where

$$G' = (V - \{x, y\} \cup \{v_{x,y}\},\ E - E_G(x) - E_G(y) \cup \{\{v_{xy}, z\} \mid z \in N_G(x, y)\}),$$

where $v_{xy} \notin V$ is a new vertex, not in G. In this case we say that G' *occurs from* G *after an edge contraction*. If H occurs from G after a sequence of edge removals or contractions, we say that H is a *minor* of G, and that G is a *major* of H.

Given a graph H, a family of graphs $\mathcal{G}$ is *H-minor-free* is no graph in $\mathcal{G}$ contains H as a minor. For instance, planar graphs are K_5-minor-free and $K_{3,3}$-minor-free due to Kuratowski Theorem [42, 95].

Graph minors have been the topic of the so-called *graph minor theory*. This deep theory, mainly developed by Robertson and Seymour in a long series of papers, describes the structure of graphs with excluded minors. It culminates in the Graph Minors Theorem [178],

which states that every class of graphs closed under taking minors can be characterized by a finite set of excluded minors. Equivalently, it says that graphs are well-quasi ordered (WQO) by the minor relation, which means that in every infinite sequence or graphs there are two of them such that one is a minor of the other. The theory also has significant algorithmic consequences. Robertson and Seymour [176] proved that every class of graphs that is closed under taking minors (this means that if a graph G is inside the class, so is every minor of G) can be recognized in cubic time.

I.2 Computational Complexity

We provide in this section some basic definitions concerning basic classes, approximation algorithms and hardness of approximation to be freely used throughout this thesis. We assume that the reader is familiar with the classes P and NP, as well as with the asymptotic notation (like O, o, Ω, or Θ). For additional background material, the reader is referred to the books of Garey and Johnson [132] and Varizani [191].

I.2.1 Approximation algorithms

Given an NP-hard minimization (resp. maximization) problem Π and a polynomial time algorithm $\mathcal{A}$, let $OPT_\Pi(I)$ be the optimal value of the problem Π for the instance I, and let $ALG(I)$ be the value given by algorithm $\mathcal{A}$ for the instance I. We say that $\mathcal{A}$ is an *α-approximation algorithm* for Π if for any instance I of Π, $OPT_\Pi(I)/ALG(I) \geq \alpha$ (resp. $OPT_\Pi(I)/ALG(I) \leq \alpha$).

The class Apx consists of all NP-hard optimization problems that can be approximated within a constant factor. The subclass PTAS (Polynomial Time Approximation Scheme) contains the problems that can be approximated in polynomial time within a ratio $1+\varepsilon$ for *any* constant $\varepsilon > 0$. Assuming P $\neq$ NP, there is a strict inclusion of PTAS in Apx (for instance, Vertex Cover is in Apx \ PTAS), hence an Apx-hardness result for a problem implies the non-existence of a PTAS.

I.2.2 Hardness of approximation

For the inapproximability results presented in this thesis, we make use of the following reductions (c.f. for instance [191]).

For two minimization problems Π_1 and Π_2, a *gap-preserving reduction* from Π_1 to Π_2, parameterized by (f_1,α) and (f_2,β), where $\alpha,\beta : \mathbb{N}^+ \to \mathbb{N}^+$ and $f_1, f_2 : \{0,1\}^* \to \mathbb{R}^+$, is a procedure that given an instance x of Π_1, computes in polynomial time an instance y of Π_2 such that:

- if $OPT(x) \leq f_1(x)$, then $OPT(y) \leq f_2(x)$.
- if $OPT(x) > \alpha(|x|)f_1(x)$, then $OPT(y) > \beta(|x|)f_2(x)$.

The usefulness of gap-preserving reductions stems from the fact that if there is a gap-preserving reduction from Π_1 to Π_2 and it is NP-hard to approximate Π_1 within a factor strictly less than α, then it is also NP-hard to approximate Π_2 within a factor strictly less than β.

In general, inapproximability results are harder to proof than just NP-hardness. In particular, the existence of a PTAS is a difficult question to answer in general. For instance, in the case of the DENSE k-SUBGRAPH PROBLEM, whose best approximation ratio is $O(n^{\delta})$ for some $\delta < 1/3$ [112], the non-existence of a PTAS has been proved recently involving very technical details [152].

I.2.3 Parameterized complexity

Parameterized complexity is a recent approach to deal with intractable computational problems having some parameters that can be relatively small with respect to the input size. This area has been developed extensively during the last decade. The monograph of Downey and Fellows [101] provides a good introduction, and for recent developments see the books by Flum and Grohe [121] and by Niedermeier [166].

A *parameter* $\mathbf{P}$ is any function mapping graphs to non-negative integers. Examples of parameters are the size of a minimum vertex cover or the size of a maximum clique. The *parameterized problem* associated with parameter $\mathbf{P}$ asks, for some fixed k, whether $\mathbf{P}(G) \geq k$ for a given graph G.

For decision problems with input size n and parameter k, the goal is to design an algorithm with running time $f(k) \cdot n^{O(1)}$, where f depends only on k. Problems having such an algorithm are said to be *fixed-parameter tractable* (FPT). There is also a theory of parameterized intractability to identify parameterized problems that are unlikely to admit fixed-parameter tractable algorithms. There is a hierarchy of intractable parameterized problem classes above FPT, the most important ones being:

$$FPT \subseteq M[1] \subseteq W[1] \subseteq M[2] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq XP$$

The principal analogue of the classical intractability class NP is $W[1]$, which is a strong analogue, because a fundamental problem complete for $W[1]$ is the k-STEP HALTING PROBLEM FOR NONDETERMINISTIC TURING MACHINES (with unlimited nondeterminism and alphabet size); this completeness result provides an analogue of Cook's Theorem in classical complexity. A convenient source of $W[1]$-hardness reductions is provided by the result stating that k-CLIQUE is complete for $W[1]$. The principal "working algorithmic" way of showing that a parameterized problem is unlikely to be fixed-parameter tractable, is to prove its $W[1]$-hardness using a parameterized reduction, which is defined as follows.

Let Π, Π' be two parameterized problems, with instances (x, k) and (x', k'), respectively. We say that Π is (uniformly many:1) *reducible* to Π' if there is a function Φ, called a *parameterized reduction*, which transforms (x, k) into $(x', g(k))$ in time $f(k) \cdot |x|^{\alpha}$, where $f, g : \mathbb{N} \to \mathbb{N}$ are arbitrary functions and α is a constant independent of k, so that $(x, k) \in \Pi$ if and only if $(x', g(k)) \in \Pi'$.

The notions of *kernel* and *kernelization* play a fundamental role in parameterized complexity. It captures data reduction taken from a fixed-parameter complexity point of view. Let $\mathcal{L}$ be a parameterized problem consisting of input (I,k), where I is the problem instance and k is the parameter. The term *reduction to a problem kernel* or *kernelization* means replacing instance (I,k) by a "reduced" instance (I',k') (called *problem kernel*) such that

1. $k' \leq k$ and $|I'| \leq g(k)$ for some function g only depending on k;
2. $(I,k) \in \mathcal{L}$ if and only if $(I',k') \in \mathcal{L}$; and
3. the reduction from (I,k) to (I',k') is computable in polynomial time in both $|I|$ and k.

The importance of kernels comes from the fact that a parameterized problem is fixed-parameter tractable if and only if it has a kernelization [101, 121, 166].

Minor closed parameters. We say that a parameter $\mathbf{P}$ is *minor closed* if whenever H is a minor of G, $\mathbf{P}(H) \leq \mathbf{P}(G)$. Examples of minor closed parameters are the size of a longest path or the size of a minimum feedback vertex set. A powerful algorithmic consequence of the Graph Minors Theorem [178] is that every minor closed parameterized problem is in FPT, i.e., it admits an algorithm running in $O(f(k) \cdot n^{O(1)})$ time. As we will discuss in more detail in Chapters 6-8, the drawback of this result is that the function $f(k)$ and the constants hidden in the big-Oh notation can be huge, and therefore these general FPT algorithms can be of limited practical value.

I.2.4 Some classical problems

For the sake of completeness, we provide here the definition of some classical problems (all NP-hard except Maximum Matching that is in P) that are mentioned in this thesis. For a complete list of classical NP-hard optimization problems, we refer the reader to [85, 132].

Maximum Matching
Input: A graph $G = (V,E)$.
Output: A subset $E' \subseteq E$ of the maximum size such that no two edges in E' share a common endpoint.

Maximum Clique
Input: A graph $G = (V,E)$.
Output: A subset $S \subseteq V$ of the maximum size such that there is an edge in E between any two vertices in S.

Maximum Independent Set
Input: A graph $G = (V,E)$.
Output: A subset $S \subseteq V$ of the maximum size such that there is no edge in E between any two vertices in S.

MINIMUM VERTEX COLORING
Input: A graph $G = (V, E)$.
Output: A function $f : V \to \{1, 2, \ldots, c\}$ such that $f(u) \neq f(v)$ for each edge $\{u, v\} \in E$ and such that c is minimized.

MINIMUM EDGE COLORING
Input: A graph $G = (V, E)$.
Output: A function $f : E \to \{1, 2, \ldots, c\}$ such that $f(e) \neq f(e')$ whenever e and e' share an endpoint and such that c is minimized.

MINIMUM VERTEX COVER
Input: A graph $G = (V, E)$.
Output: A subset $S \subseteq V$ of the minimum size such that for every edge $e = \{u, v\} \in E$, either $u \in S$ or $v \in S$.

MINIMUM FEEDBACK VERTEX SET
Input: A graph $G = (V, E)$.
Output: A subset $S \subseteq V$ of the minimum size such that $G[V \setminus S]$ has no cycles.

MINIMUM DOMINATING SET
Input: A graph $G = (V, E)$.
Output: A subset $S \subseteq V$ of the minimum size such that for every vertex $u \in V, \setminus S$ there is a $v \in S$ such that $\{u, v\} \in E$.

LONGEST PATH
Input: A graph $G = (V, E)$.
Output: A path in G with the maximum number of edges.

LONGEST CYCLE
Input: A graph $G = (V, E)$.
Output: A cycle in G with the maximum number of edges.

DENSE k-SUBGRAPH (DkS)
Input: A graph $G = (V, E)$.
Output: A subset $S \subseteq V$, with $|S| = k$, such that $\rho(S)$ is maximized.

MINIMUM SET COVER
Input: A collection $\mathcal{C}$ of subsets of a finite set S.
Output: A subset $\mathcal{C}' \subseteq \mathcal{C}$ of the minimum size such that every element in S belongs to at least one member of $\mathcal{C}'$.

Part II

Traffic Grooming

II.1 Motivation

Optical *Wavelength Division Multiplexing* (WDM) is today the most promising technology to accommodate the explosive growth of Internet and telecommunication traffic in wide-area, metro-area, and local-area networks. Using WDM, the potential bandwidth of 50 THz of a fiber can be divided into multiple non-overlapping wavelengths or frequency channels. Since currently the commercially available optical fibers can support over a hundred frequency channels, such a channel has over one gigabit-per-second transmission speed. However, the network is usually required to support traffic connections at rates that are lower than the full wavelength capacity. In order to save equipment cost and improve network performance, it turns out to be very important to aggregate the multiple low-speed traffic connections, namely *requests*, into higher-speed streams. Traffic grooming is the generic term used to carry out this aggregation, in order to improve the usage of the bandwidth and of the components, and therefore to reduce the network cost.

When establishing a connection in an optical network, one has to install some equipments at both extremities of the connection, typically an optical transmitter (laser) at its source and an optical receiver at its destination. Due to the cost of building, installing, and maintaining devices, it is usually more interesting to use a single kind of device that may handle both transmission and reception, instead of two distinct devices. Such devices are called *Light Termination Equipment*, or LTE for short. Therefore, every connection is involved in two distinct LTEs, and two distinct connections may share the same LTE, provided that one ends at a node while the other originates at the same node. In this context, the traffic grooming problem refers to minimizing the number of LTEs that are needed in the network to serve all connection requests. The problem of minimizing the number of LTEs in the network being NP-hard [105,157], research effort has concentrated on the development of efficient approximation algorithms for both static and on-line traffic [86,108,119,120,182].

At another level of the network, traffic grooming refers in a more general sense to techniques used to combine low-speed traffic streams onto high-speed wavelengths in order to minimize the network-wide cost in terms of electronic switching. In this part of the thesis we focus on this version of traffic grooming. Nodes of the network insert and/or extract the data streams on a wavelength by means of *Add-Drop Multiplexers* (ADMs for short). A WDM optical network can handle many wavelengths, each with large bandwidth available (nowadays, in the order of 40 Gbps). On the other hand, a single user seldom usually does not need such large bandwidth. Therefore, by using multiplexed access such as *Time-Division Multiple Access* (TDMA) or *Code-Division Multiple Access* (CDMA), different users can share the same wavelength, thereby optimizing the bandwidth usage of the network. By using traffic grooming, not only the bandwidth usage is optimized, but also (and more importantly) the cost of the network can be cut by reducing the total number of ADMs. Such techniques become increasingly important for emerging network technologies, including SONET/WDM rings and MPLS/MPλS backbones [186], for which traffic grooming is essential.

In this context, one ADM is needed in a node each time we want to add or drop traffic from a wavelength at this node. Therefore, one has to place one ADM in a node for each wavelength in which traffic is added or dropped, as it is illustrated in Figure II.1. Here,

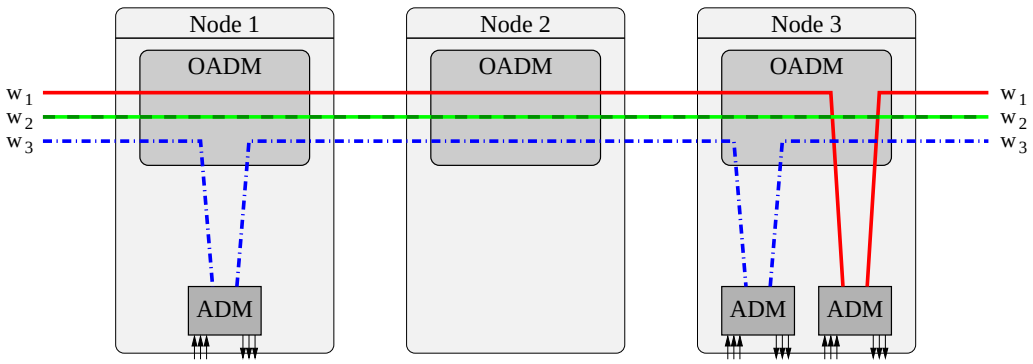

Figure II.1: Placement of ADMs in the network: one ADM for each wavelength used in a node.

the bandwidth requirement of a traffic stream is expressed as a fraction of the bandwidth offered by a single wavelength, which is called the *grooming factor* and henceforth denoted by C. Hence, an ADM is able to drop (resp. add) up to C unitary traffic streams from (resp. to) a given wavelength. Thus, the traffic grooming problem consists in minimizing the total number of ADMs to be installed in the network in order to accommodate all traffic streams.

The general traffic grooming problem with respect to the traffic requirement being NP-hard [69], recent works focus on specific issues. Most of the existing algorithms aim at grooming traffic in such a way that all the traffic between any given pair of nodes is carried on a minimum number of wavelengths. However, a large part of the network cost depends on the capacity of the multiplexing equipment required at each node. Hence, in order to minimize the overall network cost, algorithms have to take into account a trade-off between the number of wavelengths used and the number of required ADMs. Indeed, minimizing the number of ADMs may be incompatible with minimizing the number of wavelengths: the number of wavelengths and the number of ADMs cannot always be simultaneously minimized (see [50, 69, 133] for examples with unitary traffic). Both minimization problems have been considered by many authors. See for example the surveys [40, 103] for minimization of the number of wavelengths, [48, 133, 134, 145, 192, 197] for minimization of ADMs, and [144, 154] for on-line approaches. Numerical results, heuristics, and tables might be found in [50,193]. It makes also sense to aim at minimizing the number of *Optical Add-Drop Multiplexers* (OADMs for short, see Figure II.1), which are devices that are able to insert/extract entire wavelengths to/from an optical fiber [117]. The reader may also consult the following surveys [74, 104, 162, 198] and books [102, 184, 200] for other aspects of traffic grooming that are not considered here.

The sequel of this introduction to traffic grooming is structured as follows. We start in Section II.2 with a general definition of the traffic grooming problem and some examples. In Section II.3 we give an overview of the variants of traffic grooming that have been considered in this thesis, as well as our main contributions to each of these variants.

II.2 Problem Definition and Examples

We first give a precise description of LTE and ADM, and then we formalize the traffic grooming problem.

A Light Termination Equipment (LTE) is a device that realizes the interface between the optical and electronic domains. It is constituted of one optical receiver and one optical transmitter, so every connection involves two disctinct LTEs, one at each endpoint. We assume that the receiver and the transmitter of a LTE are tuned on the same wavelength (we would like to stress that other assumptions are technologically possible). Also, two distinct connections may share a LTE, provided that one ends at a node while the other originates at the same node, and that both connections are assigned the same wavelength.

An Add-Drop Multiplexer (ADM) is a device used in synchronous transmission networks (like SDH or SONET, see [102]) to add (insert) or drop (remove) lower-rate traffic channels from a higher-rate aggregated channel. In optical networks, each ADM contains a LTE to realize the interface between the optical domain (high-speed channel) and the electronic domain (lower-speed channels). Thus an ADM operates on a single high-speed data stream and therefore on a single wavelength, as can be seen in Figure II.1. The cost of an ADM is given by its capacity, that is, the maximum number of low-speed channels (provided that each of them have unitary bandwidth requirement) that can be added or dropped from the wavelength. The capacity of an ADM is called the grooming factor or grooming ratio C. Finally, remark that a LTE is a special case of an ADM for $C = 1$.

In optical networks with grooming capabilities, the traffic demands are expressed in terms of low-speed data channels. Thus, one has to assign to each connection request a path and a wavelength while respecting that at most C connection requests can be assigned the same wavelength on the same link of the network.

The precise statement of the traffic grooming problem depends on the particular assumptions, like the considered topology (for instance, a path or a ring) or the traffic pattern (for instance, an all-to-all setting or a bounded-degree request graph). This is the reason why we shall state the precise definition of each considered model in the corresponding chapter. For the sake of intuition, we provide here a general statement that does not capture the particularities of each setting.

A general instance of the Traffic Grooming problem is a triple (G, R, C), where $G = (V, E)$ is a digraph modeling the network topology, R is a set of connection requests and C is a positive integer, namely the grooming factor. Given a connection request $r \in R$ identified by a couple of nodes aiming to communicate, let P_r be the set of the directed paths in G connecting the two endpoints relative to r. There are two main issues to be addressed:

- the determination of a path system (or path assignment) of (G, R), that is a function $p : R \mapsto \bigcup_{r \in R} P_r$;
- the determination of a proper wavelength assignment of (G, R), that is a function $w : R \mapsto \mathbb{N}^+$ such that for any arc $e \in E$ at most C paths using e are assigned the same wavelength.

Every request $r \in R$ needs an ADM at each of its endpoint nodes. The key point is that the same ADM can be shared by the paths having a common endpoint which are assigned the same wavelength. The traffic grooming problem is the optimization problem of finding functions p, w for (G, R, C) minimizing the total number of used ADMs.

As we shall see in Chapters 1-4, the results presented in this thesis deal mainly with the second issue, that is, the assignment w of wavelengths to the requests. To fix ideas we provide now two examples, for unidirectional and bidirectional rings, respectively.

Unidirectional ring. Suppose we have a unidirectional ring with 4 nodes $\{1, 2, 3, 4\}$ and an all-to-all symmetric unitary traffic (i.e., one request between each couple of nodes). When the traffic requirement is symmetric, it can be easily shown (by exchanging wavelengths) that there always exists an optimal solution in which the same wavelength is given to each pair of symmetric requests. Thus without loss of generality we assign to each pair of symmetric requests, called a *circle*, the same wavelength. Then each circle uses $\frac{1}{C}$ of the bandwidth of the whole ring. If the two end-nodes of a circle are i and j, we need one ADM at node i and one at node j. The main point is that if two requests have a common end-node, they can share an ADM if they are assigned the same wavelength. In our example, there are therefore such 6 circles (i, j) for $1 \leq i < j \leq 4$. If there is no grooming (i.e., $C = 1$) we need 6 wavelengths (one per circle) and a total of 12 ADMs. If we have a grooming factor $C = 2$, we can put on the same wavelength two circles, using 3 or 4 ADMs according to whether they share an end-node or not. For example, we can put together $(1, 2)$ and $(2, 3)$ on one wavelength; $(1, 3)$ and $(3, 4)$ on a second wavelength, and $(1, 4)$ and $(2, 4)$ on a third one, for a total of 9 ADMs, and this is optimal. Now, if we allow a grooming factor $C = 3$, we can use only 2 wavelengths. Indeed, if we put together on one wavelength $(1, 2)$, $(2, 3)$, and $(3, 4)$ and on the other one $(1, 3)$, $(2, 4)$, and $(1, 4)$ we need 8 ADMs (first solution in Figure II.2); but we can do better by putting on the first wavelength $(1, 2)$, $(2, 3)$, and $(1, 3)$ and on the second one $(1, 4)$, $(2, 4)$, and $(3, 4)$, therefore using 7 ADMs (second solution in Figure II.2).

More formally, in the above example with 4 vertices and $C = 3$, the first solution consists in a partition of the edges of K_4 (each edge of K_4 corresponds to a circle) into two paths with four vertices each: $[1, 2, 3, 4]$ and $[1, 4, 2, 3]$, while the second solution corresponds to a decomposition into a triangle $(1, 2, 3)$ and a star with edges $(1, 4)$, $(2, 4)$, and $(3, 4)$ (see Figure II.2).

Bidirectional ring. Consider now a bidirectional ring on five nodes $\{0, 1, 2, 3, 4\}$ with all-to-all unitary traffic modeled by the complete symmetric digraph K_5^*. In this setting, it is better (in terms of the number of used ADMs) to route requests (i, j) and (j, i) on different wavelengths using shortest path routing. For example, with grooming factor $C = 3$, we can put on one wavelength the requests $\{(i, i + 1 \bmod 5), (i, i + 2 \bmod 5) : i = 0, \ldots, 4\}$ routed clockwise, and on another wavelength the requests $\{(i, i - 1 \bmod 5), (i, i - 2 \bmod 5) : i = 0, \ldots, 4\}$ routed counterclockwise. We need 5 ADMs on each wavelength, so overall 10 ADMs. But if requests (i, j) and (j, i) are routed on a same wavelength, then we can put at most 3 circles (pairs of symmetric requests) per wavelength, using at least 3 ADMs. Since K_5^* contains 10 circles, we need at least 4 wavelengths: 3 of them with 3 circles each

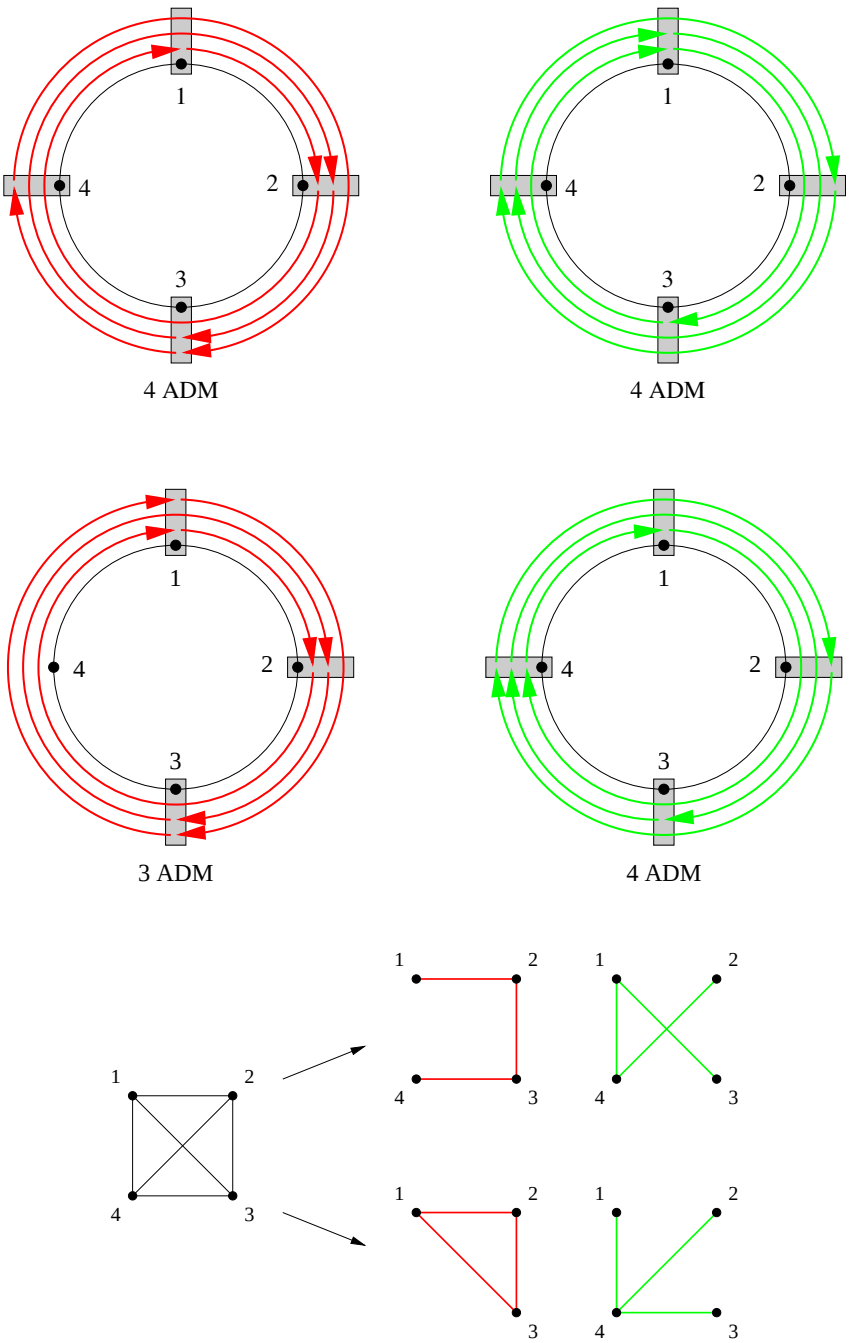

Figure II.2: Traffic grooming for a unidirectional ring with 4 nodes, grooming factor $C = 3$, and all-to-all unitary traffic. The above solution uses $4+4 = 8$ ADMs, whereas the second one uses $3+4 = 7$ ADMs. Below, the corresponding partitions of K_4 are illustrated.

(and therefore at least 3 ADMs each) and one of them with at least 1 circle and 2 ADMs, so overall at least 11 ADMs.

With grooming factor $C = 2$, we can put on one wavelength requests $\{(i, i+1 \bmod 5) : i = 0, \ldots, 4\}$ and on another wavelength requests $\{(i, i+2 \bmod 5) : i = 0, \ldots, 4\}$. Symmetric requests are routed similarly in the opposite direction. We obtain the first partition of Figure II.3, using overall $2 \cdot 10 = 20$ ADMs. But we can do better by putting on a first wavelength requests $\{(i, i+1 \bmod 5) : i = 0, \ldots, 4\}$, request $(0, 2)$, and request $(2, 4)$ using 5 ADMs, and on a second wavelength requests $(1, 3)$, $(3, 5)$, and $(4, 1)$ using 4 ADMs. We obtain the second partition of Figure II.3, using $2 \cdot (5 + 4) = 18$ ADMs overall.

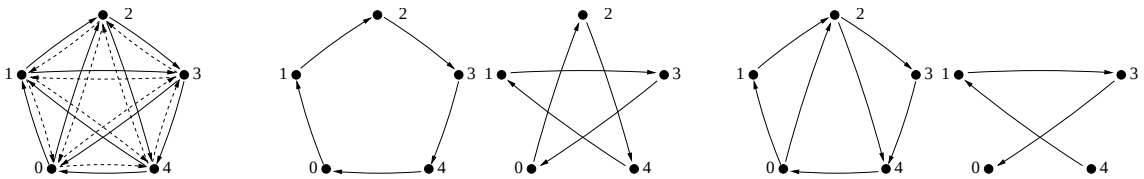

Figure II.3: On the left, a K_5^*. In the middle and on the right, two valid partitions of K_5^* when $C = 2$ using 10 and 9 ADMs, respectively. Symmetric requests are routed counterclockwise and partitioned similarly, hence using 20 and 18 ADMs, respectively.

II.3 State-of-the-art and our Contribution

The topic of traffic grooming is that large that it would be too ambitious to pretend to provide here a complete state-of-the-art. Most of the literature about traffic grooming originates from an engineering context (cf. for instance the books and surveys [B4, 74, 102, 104, 162, 184, 198, 200]), hence the vast majority of articles about traffic grooming propose heuristics that are then contrasted by the corresponding simulations. In this thesis we are only concerned with the theoretical aspects of traffic grooming, such as complexity results, approximation algorithms with a provable approximation ratio, or the construction of optimal solutions for restricted instances and topologies. It is far from the intention of the author to suggest that purely theoretical results are more important or more significant than applied ones; the fact that this thesis does not cover the literature originating from the engineering community is just a matter of scientific knowledge and interests.

We survey the existing results and contextualize our contributions concerning hardness and approximation, the all-to-all set of requests, and pseudo-dynamic scenarios in Sections II.3.1, II.3.2, and II.3.3, respectively.

II.3.1 Hardness and approximation

Most approximation algorithms and hardness results deal with the case where the topology is a ring or a path. This is because, on the one hand, these topologies are widely used in practical applications (like SONET WDM rings). On the other hand, whereas the traffic grooming problem can be clearly stated in paths and rings (see Chapter 1), the notion of

traffic grooming in topologies with vertices of degree at least three is somehow ambiguous, as it heavily depends on the technological devices used in each network. Approximation algorithms and NP-hardness results for a simplified model of traffic grooming in stars and trees can be found in [116]. Another model of traffic grooming in stars and trees is studied in [146]. From now on we focus on the case when the physical topology is a ring or a path.

Hardness results. The notion of traffic grooming with $C > 1$ was introduced in [134] for the ring topology. The problem has been proved to be NP-hard for ring networks and general C [69] using a reduction from the Bin Packing problem. Another proof was also mentioned in [192]. On the other hand, there was no result on the inapproximability of the problem for fixed $C \geq 1$. In [72] Chow and Lin conjectured that Ring Traffic Grooming is Max SNP-hard (or equivalently, Apx-hard, modulo PTAS-reductions) for any fixed value of the grooming factor. We answer affirmatively to this question in Theorem 1.3 of Chapter 1, providing the first hardness result for the Ring Traffic Grooming problem for fixed values of the grooming factor C.

Considering C as part of the input, in [146] it was proved that Path Traffic Grooming does not accept a constant-factor approximation unless P = NP. For fixed values of C, Path Traffic Grooming was proved to be in P for $C = 1$ [43], but the complexity for fixed $C \geq 2$ has been an open question for a while. Recently, it has been proved in [181] that Path Traffic Grooming for fixed $C > 1$ is NP-hard for *bounded number of wavelengths*. Our approach permits us to improve this result by proving the Apx-hardness of Path Traffic Grooming for any fixed $C > 1$ and *unbounded* number of wavelengths (see Theorem 1.5 in page 42). That is, we rule out the existence of a PTAS for fixed values of C, unless P=NP. In particular, this extends the NP-hardness result of [181] to the case where the number of wavelengths is not bounded.

The main ingredient of our approach is the proof of the Apx-completeness (given in Section 1.2 of Chapter 1) of the problem of finding the maximum number of edge-disjoint triangles in a tripartite graph with bounded degree B: Maximum B-Bounded Edge Covering by Triangles (MECT-B for short). The proof is obtained by L-reduction from Maximum Bounded Covering by 3-Sets, which was proved to be MAX SNP-complete in [149]. A simple modification of this technique permits us to prove the Apx-completeness of finding the maximum number of edge-disjoint odd cycles of given length in a graph. This later claim is then used to extend our results to arbitrary values of C, see Sections 1.2 and 1.3 of Chapter 1.

Approximation algorithms. Since Ring Traffic Grooming and Path Traffic Grooming are NP-hard, it is natural to devise polynomial-time approximation algorithms. We first focus on the ring topology.

As we discuss in Section 1.3 (page 38), it is trivial to obtain a $O(\sqrt{C})$-approximation with running time polynomial in both C and n (this fact was first proved in [136]), where n is the number of nodes of the network. For $C = 1$ (that is, for the minimization of LTEs, which is also known to be NP-hard [105,157]) the best algorithm in rings achieves an approximation ratio of 10/7 [108]. For this specific case of $C = 1$ we refer the reader to [43, 51, 108, 182].

For general C, the best approximation algorithm [118] achieves an approximation factor of $O(\log C)$ by using a classical Set Cover approach, but the problem is that the running time is exponential in C (that is, $n^{O(C)}$). Since in practical applications SONET WDM rings are widely used as backbone optical networks [104,162], the grooming factor is usually greater than the size of the network, i.e., $C \geq n$. For those networks, the running time of the algorithm of [118] becomes exponential in n. Thus, it turns out to be important to find good approximation algorithms with running time polynomial in both n and C. In Section 1.4 of Chapter 1 we provide such an approximation algorithm, considering C as part of the input. Our algorithm finds a solution of Ring Traffic Grooming that approximates the optimal value within a factor $O(n^{1/3}\log^2 n)$ for any $C \geq 1$. To the best of our knowledge, this is the first polynomial-time approximation algorithm for the Ring Traffic Grooming problem with an approximation ratio which does not depend on C. Although the performance of this algorithm seems not to be very good at first sight, in fact we conjecture that for the general instance of the problem it is not possible to get rid of a factor n^{δ}, for some constant $\delta > 0$ (see Conjecture 1.1 in page 47). We also show that the general scheme of the algorithm yields a $O(\log^2 n)$-approximation if the request graph excludes a fixed graph as minor, for example if it is planar or of bounded genus. The main theoretical contribution of this algorithm is to relate the Traffic Grooming problem to the widely studied Dense k-Subgraph problem [112].

Concerning the case of the path, both the algorithm of [118] (with approximation ratio $O(\log C)$ fox fixed values of C) and the algorithm we present in Chapter 1 (with approximation ratio $O(n^{1/3}\log^2 n)$ irrespective of C) also apply to the path topology with slight modifications.

II.3.2 The all-to-all case

An important special case is given by a unidirectional SONET ring with n nodes, grooming ratio C, and all-to-all uniform unitary traffic. This problem has been modeled as a graph partitioning problem in both [47] and [136]. In the all-to-all case the set of requests is modeled by the complete graph K_n. To a wavelength λ is associated a subgraph B_λ in which each edge corresponds to a pair of symmetric requests (that is, a circle in the terminology used in Section II.2) and each node to an ADM. The grooming constraint, i.e., the fact that a wavelength can carry at most C requests, corresponds to the fact that the number of edges $|E(B_\lambda)|$ of each subgraph B_λ is at most C. The cost corresponds to the total number of vertices used in the subgraphs, and the objective is therefore to minimize this number.

Traffic Grooming in the Unidirectional Ring with All-to-all Requests
Input: Two integers n and C.
Output: Partition $E(K_n)$ into subgraphs B_λ, $1 \leq \lambda \leq \Lambda$, s.t. $|E(B_\lambda)| \leq C$ for all λ.
Objective: Minimize $\sum_{\lambda=1}^{\Lambda} |V(B_\lambda)|$.

With this setting, optimal constructions have been obtained (using tools of graph and design theory [75]) for the cases $C = 3$ [44], $C = 4$ [47, 145], $C = 5$ [46], $C = 6$ [45],

$C = 7$ [76], and $C \geq n(n-1)/6$ [50]. Recently, good approximated solutions for any value of C have been presented in [49]. The all-to-all traffic case has been also studied on the path in [43]. See [48] for a survey of most of these results.

Nevertheless, the case when the physical network is a bidirectional ring has been much less studied in the literature from this graph partitioning perspective, mostly because the above simplified model cannot be applied anymore. A MILP formulation of the problem can be found in [145]. In [82] tools from design theory were applied to the bidirectional ring for the special case $C = 8$, without providing lower bounds to compare the proposed solutions to the optimal ones. Finally, a lower bound (independent of the routing of the requests) was given in [72].

In a bidirectional ring, requests are routed either clockwise or counterclockwise. We study in Chapter 3 the bidirectional ring with symmetric shortest path routing and all-to-all traffic. This is the first attempt to deal with traffic grooming in bidirectional rings using an approach similar to [48]. Using graph partitioning techniques and combinatorial designs, we formally state the problem, provide general lower bounds, and construct infinite families of optimal solutions for $C \in \{1, 2, 3\}$ and C of the form $k(k+1)/2$, as well as asymptotically optimal solutions. See Chapter 3 for the details.

II.3.3 Pseudo-dynamic scenarios

In this section we discuss two variants of the traffic grooming in order to allow more dynamic settings than the ones discussed so far. Chapters 2 and 4 are concerned with a changeable request graph and a changeable grooming factor, respectively. We observe that these scenarios are not completely dynamic, in the sense of considering any request graph and any grooming factor, but they incorporate a flexibility that did not exist in the theoretical models considered so far.

Variable grooming factor. Most of the papers on grooming deal with a single (static) traffic matrix. Some articles consider variable (dynamic) traffic, such as finding a solution which works for the maximum traffic demand [55,199], but all keep a fixed grooming factor. Recently, an interesting variation of the traffic grooming problem, called grooming for two-period optical networks, has been introduced by Colbourn, Quattrocchi, and Syrotiuk in [78, 79] in order to capture some dynamic nature of the traffic. Informally, in the two-period grooming problem each time period supports different traffic requirements. During the first period of time there is all-to-all uniform traffic among n nodes, each request using $1/C$ of the bandwidth; but during the second period there is all-to-all traffic only among a subset V of v nodes, each request now being allowed to use a larger fraction of the bandwidth, namely $1/C'$ where $C' < C$.

In [78, 79] the authors completely solved the cases when $C = 2$ and $C = 3$ ($C' = 1$ or 2). In Chapter 4 we determine the minimum drop cost for all $n \geq v \geq 0$, $C = 4$, and $C' \in \{1, 2, 3\}$. If we further restrict the solution to use the minimum number of wavelengths, it turns out that the optimal drop cost under this constraint may differ from the absolute optimum (see the last paragraph before Section 4.5 of Chapter 4). We also determine the optimal

drop cost that uses the minimum number of wavelengths for all $n \geq v \geq 0$, $C = 4$, and $C' \in \{1, 2, 3\}$. The precise cost formulas can be found in Section 4.1 of Chapter 4.

Variable request graph. As mentioned above, most of previous work on traffic grooming has focused on the case where the requests are given as input [J2, 47, 48, 104, 116, 118, 136, 162]. We consider in Chapter 2 the case where only the network topology is given, together with a bound Δ on the degree of the request graph. We would like to place, for each value of the grooming factor C, a minimum number of ADMs at each node in such a way that they could support *any* traffic pattern where each node is the end-node of at most Δ requests. This model is interesting because the network can support dynamic traffic without replacement of the ADMs. In other words, instead of placing the ADMs *a posteriori* for a given traffic demand, we would like to place them *a priori*.

From a practical point of view, it is interesting to design a network being able to support any request graph with maximum degree not exceeding a given constant. This situation is usual in real optical networks, since due to technology constraints the number of allowed communications for each node is usually bounded. This flexibility can also be thought from another point of view: if we have a limited number of available ADMs to place at the nodes of the network, then it is interesting to know which is the maximum degree of a request graph that our network is able to support, depending on the grooming factor. Equivalently, given a maximum degree and a number of available ADMs, it is useful to know which values of the grooming factor the network is able to support.

The aim of Chapter 2 is to provide a theoretical framework to design such networks with dynamically changing traffic. More precisely, we study the case when the physical network is given by a unidirectional ring. We show that the problem is essentially equivalent to finding the least integer $M(C, \Delta)$ such that the edges of any graph with maximum degree at most Δ can be partitioned into subgraphs with at most C edges and each vertex appears in at most $M(C, \Delta)$ subgraphs (see Section 2.1 for the details). We establish the value of $M(C, \Delta)$ for almost all values of C and Δ, leaving open only the case where $\Delta \geq 5$ is odd, $\Delta \pmod{2C}$ is between 3 and $C - 1$, $C \geq 4$, and the request graph does not contain a perfect matching (see Table 2.1 in page 65 for a summary of the results). For these open cases, we provide upper bounds that differ from the optimal value by at most one.

Chapter 1

Hardness and Approximation

In this chapter we focus on traffic grooming on ring and path topologies. On the one hand, we provide an inapproximability result for Traffic Grooming for fixed values of the grooming factor C, answering affirmatively to a conjecture of Chow and Lin [72]. More precisely, we prove that Ring Traffic Grooming for fixed $C \geq 1$ and Path Traffic Grooming for fixed $C \geq 2$ are Apx-complete, even if the maximum degree of the request graph is bounded by a small constant. That is, they do not accept a PTAS unless P = NP. Both results rely on the fact that finding the maximum number of edge-disjoint triangles in a tripartite graph (and more generally cycles of length $2C + 1$ in a $(2C + 1)$-partite graph of girth $2C + 1$) is Apx-complete.

On the other hand, we provide a polynomial-time approximation algorithm for Ring and Path Traffic Grooming, based on a greedy cover algorithm, with an approximation ratio independent of C. Namely, the approximation guarantee is $O(n^{1/3} \log^2 n)$ for any $C \geq 1$, n being the size of the network. This is useful in practical applications, since in backbone networks the grooming factor is usually greater than the network size. Finally, we improve this approximation ratio under some extra assumptions about the request graph.

Keywords: traffic grooming, optical networks, SONET ADM, approximation algorithms, Apx-hardness, PTAS.

1.1 Introduction

As already mentioned in Section II.1 (page 23), the most accepted criterion to reduce the equipment cost in WDM optical networks is to minimize the number of electronic terminations, which is unanimously considered as the dominant cost, rather than the number of wavelengths. SONET ring is the most widely used optical network infrastructure today. In these networks, a communication between a pair of nodes is done via a *lightpath*, and each lightpath uses an Add-Drop Multiplexer (ADM), i.e., an electronic termination, at

each of its two endpoints. If each request uses $1/C$ of the capacity of a wavelength, C is said to be the *grooming factor*. We recall that the problem is equivalent to assigning a wavelength to each request in such a way that for any wavelength and any link of the network, there can be at most C requests using this link on this wavelength. The aim is to minimize the total number of ADMs.

Statement of the problem. In the graph-theoretical approach that we use, the set of requests is modeled by a graph R, and each vertex in the subgraph of R corresponding to a wavelength represents an ADM. The problem, in the case where the communication network is a ring, can be formally stated as follows.

Ring Traffic Grooming
Input: A cycle C_n on n vertices (network), a graph R (set of requests) on vertices of C_n, and a grooming factor C.
Output: Find for each edge $r = \{x, y\}$ of R, a path $P(r)$ in C_n between x and y, and a partition of the edges of R into subgraphs R_λ, $1 \leq \lambda \leq \Lambda$, such that for each edge e in $E(C_n)$ and for all λ, the number of paths $P(r)$ using e, r being an edge of R_λ, is at most C.
Objective: Minimize $\sum_{\lambda=1}^{\Lambda} \lvert V(R_\lambda)\rvert$.

The number of paths $P(r)$ using an edge $e \in E(C_n)$ in a given subgraph R_λ is known as the *load* of e in R_λ. That is, the load of the edges in any subgraph of the partition of $E(R)$ can be at most C. The statement of Path Traffic Grooming is analogous, replacing *cycle* C_n with *path* P_n. To fix ideas, consider a ring on five nodes and the complete graph of Figure 1.1 as request graph, and let $C = 2$. We exhibit two valid solutions of the problem, both using two subgraphs (i.e., two wavelengths). The lower solution is better because it uses 9 vertices instead of 10.

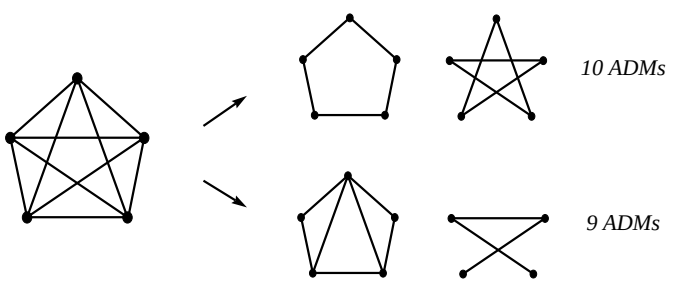

Figure 1.1: Two valid partitions of K_5 when $C = 2$, using different number of ADMs.

Our contribution. As discussed in Section II.3.1 (page 28), there was no result in the literature on the inapproximability of Ring Traffic Grooming for fixed $C \geq 1$. In [72] Chow and Lin conjectured that Traffic Grooming is Max SNP-hard (or equivalently, Apx-hard, modulo PTAS-reductions) for any fixed value of the grooming factor. We

answer affirmatively to this question in Theorem 1.3, providing the first hardness result for the Ring Traffic Grooming problem for fixed values of the grooming factor C.

Considering C as part of the input, in [146] it was proved that Path Traffic Grooming does not accept a constant-factor approximation unless P = NP. For fixed values of C, Path Traffic Grooming was proved to be in P for $C = 1$ [43], but the complexity for fixed $C \geq 2$ has been an open question for a while. Recently, it has been proved in [181] that Path Traffic Grooming for fixed $C > 1$ is NP-hard for *bounded number of wavelengths*. Our method permits us to improve this result in Section 1.3, by proving the Apx-hardness of Path Traffic Grooming for any fixed $C > 1$ and *unbounded* number of wavelengths. In particular, this extends the NP-hardness result of [181] to the case where the number of wavelengths is not bounded.

The main ingredient of our approach is the proof of the Apx-completeness (given in Section 1.2) of the problem of finding the maximum number of edge-disjoint triangles in a tripartite graph with bounded degree B: Maximum B-Bounded Edge Covering by Triangles (MECT-B for short). The proof is obtained by L-reduction from Maximum Bounded Covering by 3-Sets, which was proved to be MAX SNP-complete in [149]. A simple modification of this technique permits us to prove the Apx-completeness of finding the maximum number of edge-disjoint odd cycles of given length in a graph. This later claim is then used to extend our results to arbitrary values of C, see Sections 1.2 and 1.3.

The design of approximation algorithms for Traffic Grooming is the topic of the second part of this chapter. We present the results for the ring topology, but the same algorithm works also for the path topology. As we show in Section 1.3, it is trivial to obtain a $O(\sqrt{C})$-approximation with running time polynomial in C and n. For $C = 1$, the best algorithm in rings achieves an approximation ratio of 10/7 [108]. For general C, the best approximation algorithm [118] achieves an approximation factor of $O(\log C)$, but the problem is that the running time is exponential in C (that is, $n^{O(C)}$). Since in practical applications SONET WDM rings are widely used as backbone optical networks [104, 162], the grooming factor is usually greater than the size of the network, i.e., $C \geq n$. For those networks, the running time of this algorithm becomes exponential in n. Thus, it turns out to be important to find good approximation algorithms with running time polynomial in both n and C. In Section 1.4 we provide such an approximation algorithm, considering C as part of the input. Our algorithm finds a solution of Ring Traffic Grooming that approximates the optimal value within a factor $O(n^{1/3} \log^2 n)$ for any $C \geq 1$. To the best of our knowledge, this is the first polynomial-time approximation algorithm for the Ring Traffic Grooming problem with an approximation ratio which does not depend on C. Although the performance of this algorithm seems not to be very good at first sight, in fact we conjecture that for the general instance of the problem it is not possible to get rid of a factor n^{δ}, for some constant $\delta > 0$. Finally, we show that the general scheme of the algorithm yields a $O(\log^2 n)$-approximation if the request graph excludes a fixed graph as a minor, for example if R is planar or of bounded genus. The main theoretical contribution of the second part of this chapter is to relate the Traffic Grooming problem to the Dense k-Subgraph problem [112]. We conclude by proposing some further research directions to better understand the complexity of Traffic Grooming.

1.2 Apx-completeness of MECT-B

The problem of finding the maximum number of node- or edge-disjoint cycles in an undirected graph G has several applications, for instance in computational biology [39]. It is often the case that both the maximum degree of G and the length of the cycles to be found are bounded by a constant. In this section we are interested in the following problem:

MAXIMUM B-BOUNDED EDGE COVERING BY TRIANGLES (MECT-B)
Input: An undirected graph G with maximum degree at most B.
Objective: Find the maximum number of edge-disjoint triangles in G.

MECT-B is long known to be NP-hard [143], and the APX-hardness when requiring node-disjoint triangles was proved in [149]. Following the ideas of [149], in [63] it was proved that MECT-5 is APX-hard for general graphs and NP-hard for planar graphs. Finally, in [160] MECT-B was studied from a parameterized view, considering the number of edge-disjoint triangles as the parameter. Namely, is was proved that MECT-B is FPT by achieving a linear kernel (see [101]).

In this chapter we prove that MECT-B remains APX-hard for tripartite graphs. For convenience, we prove the MAX SNP-hardness of MECT-B, which is known to be the same as the APX-hardness modulo PTAS-reductions [191]. MECT-B is trivially in APX, since a simple greedy algorithm provides a 3-approximation. The best approximation guarantee for MECT-B is a $(3/2+\varepsilon)$-approximation algorithm for any $\varepsilon > 0$ [147]. We need to introduce two problems to be used in the proof of Theorem 1.1: MAXIMUM BOUNDED COVERING BY 3-SETS (MAX 3SC-B for short): Given a collection of 3-subsets of a given set, each element appearing in at most B subsets, find the maximum number of disjoint subsets; and MAXIMUM BOUNDED INDEPENDENT SET (INDEP. SET-B for short): Given a graph of maximum degree $\leq B$, find a maximum independent set.

Theorem 1.1 MECT-B, $B \geq 10$, *is* APX*-complete for tripartite graphs.*

Proof: L-reduction from MAX 3SC-B and L-reduction to INDEP. SET-B.
We define h : MECT-B $\rightarrow$ INDEP. SET - (3/2(B-2)) as follows: given a graph G as instance I of MECT-B, we define the following instance $h(I)$ of INDEP. SET - (3/2(B-2)): the graph $h(G)$ contains a node v_T for every triangle T in G. There is an edge $\{v_{T_0}, v_{T_1}\}$ in $h(G)$ if and only if T_0 and T_1 share an edge in G. Given a solution A of $h(I)$, we define a solution $S_h(A)$ of I by taking the triangles corresponding to nodes in A. It is easily verified that (h, S_h) is an L-reduction.

We define f : MAX 3SC-B $\rightarrow$ MECT-(3B+1) in the following way: suppose that we are given as instance I, a collection $\mathcal{S}$ of 3-element subsets of a set X such that every element of X belongs to at most B members of $\mathcal{S}$. The problem for I consists in finding the maximal number $OPT(I)$ of disjoint subsets in $\mathcal{S}$. We construct an instance $f(I)$ of MECT-(3B+1), i.e., we construct a graph $G = (V, E)$ in which we ask for the maximum number $OPT(f(I))$ of edge-disjoint triangles. Let $\mathcal{S} = \{c_1, \ldots, c_r\}$, with $|c_i| = 3$. The local replacement f substitutes for each element $c_i = \{x, y, z\} \in \mathcal{S}$, the graph $G_i = (V_i, E_i)$ depicted in Figure 1.2.

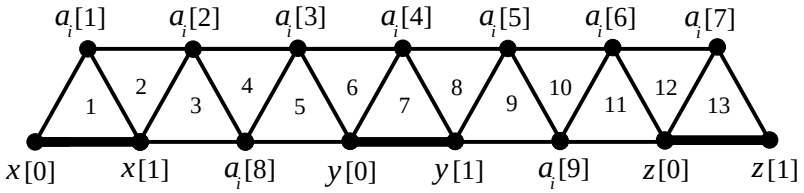

Figure 1.2: Gadget G_i used in the reduction of the proof of Theorem 1.1.

To avoid confusion, note by t any element in c_i, i.e., $t \in \{x, y, z\}$. Note that, for each element t, the nodes $t[0]$ and $t[1]$, and the edge $\{t[0], t[1]\}$ (corresponding to the thick edges in Figure 1.2) appear only once in G, regardless of the number of occurrences of t. On the other hand, we add 9 new vertices $a_i[j]$, $1 \leq j \leq 9$ for each subset c_i, $1 \leq i \leq |\mathcal{S}|$. More precisely, $G = (V, E) = \cup_{i=1}^{|\mathcal{S}|} G_i$, where $V = \bigcup_{t \in X} \{t[i] : i = 0, 1\} \cup \bigcup_{i=1}^{|\mathcal{S}|} \{a_i[j] : 1 \leq j \leq 9\}$ and $E = \bigcup_{i=1}^{|\mathcal{S}|} E_i$.

Given a solution A of $f(I)$ of size s_2, we modify it in polynomial time to another equal or better solution A' in the following way: in each G_i, if the three triangles covering the edges $\{x[0], x[1]\}$, $\{y[0], y[1]\}$, and $\{z[0], z[1]\}$ (numbered $1, 7, 13$ in Figure 1.2) belong to A, we choose the seven *odd* triangles of G_i to belong to A'. If not, we take the six *even* triangles. Let $s_2' \geq s_2$ be the size of A'. Then, we define a solution $S_f(A)$ of I by choosing the subset c_i to be in $S_f(A)$ if and only if A' contains exactly 7 triangles in G_i. We claim that the pair (f, S_f) is an L-reduction: in each G_i there are 13 different triangles, numbered from 1 to 13 in Figure 1.2. The only way to choose 7 edge-disjoint triangles in G_i is by taking all the *odd* triangles, and thus by covering the three edges $\{x[0], x[1]\}$, $\{y[0], y[1]\}$, and $\{z[0], z[1]\}$. All other choices of triangles yield at most 6 edge-disjoint triangles. The key observation is that we are able to choose 7 triangles exactly $OPT(I)$ times. Indeed, each time we choose 7 triangles we cover the edges corresponding to 3 elements of c_i, and since the number of disjoint c_i's in $\mathcal{S}$ is $OPT(I)$, this can be done exactly $OPT(I)$ times. On the other hand, one can easily see that $OPT(I) \geq \frac{|\mathcal{S}|}{3B}$. Hence:

$$\begin{aligned} OPT(f(I)) &= 7 \cdot OPT(I) + 6(|\mathcal{S}| - OPT(I)) \leq OPT(I) + 18B \cdot OPT(I) \\ &= (18B + 1)OPT(I). \end{aligned}$$

To conclude, note that if the solution $S_f(A)$ of I has size s_1, we have $OPT(I) - s_1 \leq OPT(f(I)) - s_2$. To see this, we observe that $OPT(f(I)) = 6r + OPT(I)$, and also $s_2' = 6r + s_1$, and so $OPT(f(I)) - OPT(I) = s_1 - s_2' \leq s_1 - s_2$.

Both (f, S_f) and (h, S_h) are L-reductions and MAX 3SC-B, $B \geq 3$ and Indep. Set-B, $B \geq 5$ are Max SNP-complete [149]. Thus, MECT-B, $B \geq 10$ is Max SNP-complete. Finally, note that the graph $G = (V, E)$ used in the proof is tripartite, where the vertex sets V_0, V_1, V_2 defining the tripartition are:

$$V_0 = \bigcup_{t \in X}^{|X|} t[0] \cup \bigcup_{i=1}^{|\mathcal{S}|} \{a_i[2], a_i[5]\}, \quad V_1 = \bigcup_{i=1}^{|\mathcal{S}|} \{a_i[j] : j = 1, 4, 7, 8, 9\},$$

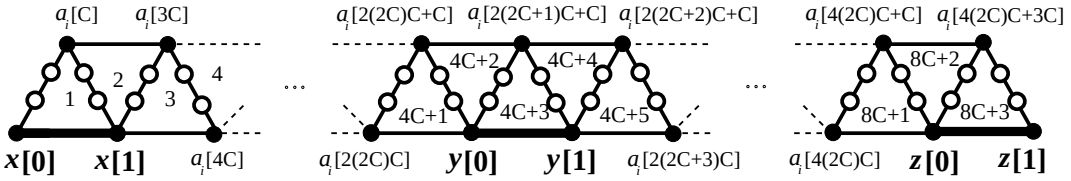

Figure 1.3: Adding $C-1$ inner points (depicted as $\circ$ in the figure) to prove the Apx-completeness of finding edge-disjoint C_{2C+1}'s.

$$V_2 = \bigcup_{i=1}^{|X|} t[1] \ \cup \ \bigcup_{t \in X}^{|\mathcal{S}|} \{a_i[3], a_i[6]\}.$$

□

The proof of the Apx-hardness of MECT-B of Theorem 1.1 can be extended to obtain the Apx-completeness of the problem of finding the maximum number of edge-disjoint cycles of length $2C+1$ for any fixed $C \geq 1$, as stated in the following theorem.

Theorem 1.2 *Let $\mathcal{G}$ be the class of $(2C+1)$-partite graphs G of girth $2C+1$, consisting of $(2C+1)$ parts $A_0, \ldots, A_{2C}$ such that the only edges are between A_i and $A_{i+1} \pmod{2C+1}$, $i = 0, \ldots, 2C$, and such that all the graphs induced by $V(G) \setminus A_i$ in G, for all $i = 0, \ldots, 2C$, form a forest. Then the problem of finding the maximum number of edge disjoint C_{2C+1}'s is* Apx*-complete in $\mathcal{G}$.*

Proof: First, note that a greedy algorithm provides a constant factor approximation with factor $2C+1$, so the problem is in Apx. Consider the gadget of the proof of Theorem 1.1 (see Figure 1.2). We modify this gadget in such a way that the same proof holds for C_{2C+1}'s instead of C_3's (triangles), and such that all the conditions of the theorem are verified. Given $C > 1$, we add a chain of $4C+1$ triangles between any two pair of triangles corresponding to *thick* edges (that is, between the edges corresponding to elements of X). Then we add $C-1$ inner points to all the edges going from up to down in the triangles. An example if shown in Figure 1.3.

It is easily seen that the graph built in this way is $(2C+1)$-partite. Indeed, it admits a partition into $(2C+1)$ parts, which consist of enumerating the vertices cyclically. Let $A_0, \ldots, A_{2C}$ be the different parts. In such a $(2C+1)$-partition, for any element $t \in X$, the vertex $t[0]$ belongs to A_0, and the vertex $t[1]$ belongs to A_{2C}. We need this property to ensure the consistency of our gadget when an element appears in more that one subset. Note that the graphs induced by $V(G) \setminus A_i$ in G, for all $i = 0, \ldots, 2C$, form a forest. At this point, one can rewrite the proof of Theorem 1.1 to obtain the result, just by changing the multiplicative constants. □

1.3 Apx-completeness of Traffic Grooming

In this section we prove the hardness results for Ring Traffic Grooming and Path Traffic Grooming. First we prove that Ring Traffic Grooming belongs to Apx

when C is fixed (i.e., not part of the input). The same result holds for Path Traffic Grooming.

Lemma 1.1 Ring Traffic Grooming *belongs to* Apx *for any fixed* $C \geq 1$.

Proof: To see that Ring Traffic Grooming is in Apx for any fixed $C \geq 1$, we have to find a constant-factor approximation algorithm. We use the fact that the best possible density ρ^* of any subgraph involved in the partition of the request graph in the ring is $O(\sqrt{C})$, given by a complete graph inducing load C in the edges of the ring (it is clear that no graph has greater density than the complete graph). We prove that the cost A of any solution $R_1, \ldots, R_\Lambda$ is in the interval $[\frac{|E(R)|}{\rho^*}, 2|E(R)|]$. This clearly implies that any solution has cost at most $2\rho^* = O(\sqrt{C})$ times the optimal cost. To see this, note that each edge of R contributes at most twice to the cost, so $A \leq 2|E(R)|$. On the other hand, we have

$$A = \sum_{\lambda=1}^{\Lambda} |V(R_\lambda)| = \sum_{\lambda=1}^{\Lambda} \frac{|E(R_\lambda)|}{\rho(R_\lambda)} \geq \sum_{\lambda=1}^{\Lambda} \frac{|E(R_\lambda)|}{\rho^*} = \frac{|E(R)|}{\rho^*}.$$

Thus, a $O(\sqrt{C})$-approximation is obtained just by taking *any* partition of the request graph. □

Since we will deal with tripartite graphs in the proof of Theorem 1.3, we need first a technical lemma concerning the structure of the optimal solutions of Ring Traffic Grooming in tripartite request graphs.

Lemma 1.2 *Let R be a tripartite instance graph of* Ring Traffic Grooming *for $C = 1$ such that the vertices belonging to the same class of the tripartition are placed consecutively in the ring, and let t^* be the maximum number of edge-disjoint triangles in R. If there exists a partition of $E(R)$ into triangles and P_4's which uses exactly t^* triangles, then this partition is optimal. The same property holds for* Path Traffic Grooming *and $C = 2$.*

Proof: We focus first on Ring Traffic Grooming. Let t^* the maximum number of edge-disjoint triangles of a partition of $E(R)$. When R is tripartite and $C = 1$, it is clear that the only possible subgraphs that can be involved in a partition of $E(R)$ are K_3, P_2, P_3, and P_4 (see Figure 1.4(a)). Since these three paths have density at most 3/4 (attained by the P_4), the cost A_t of any solution using t triangles satisfies

$$A_t \;\geq\; t + 4 \cdot \frac{|E(R)| - 3t}{3} = \frac{4}{3}|E(R)| - 3t \;\geq\; \frac{4}{3}|E(R)| - 3t^*. \tag{1.1}$$

Note that the above bound does not depend on t, and therefore holds for any solution. A partition as stated in the conditions of the lemma attains this lower bound, hence it is optimal. The same argument applies to the path and $C = 2$ (see Figure 1.4(b)), since the same subgraphs are involved in any partition. □

We are ready to state the main result of this section.

Theorem 1.3 Ring Traffic Grooming *is* Apx*-complete for fixed $C = 1$, even if the request graph has degree bounded by a constant $B \geq 10$. Thus, it does not accept a* PTAS *unless* P = NP.

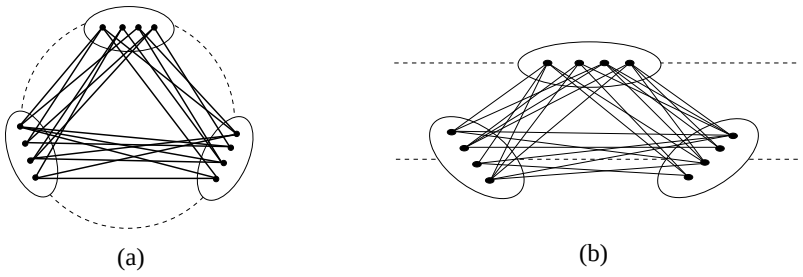

Figure 1.4: Tripartite request graphs used in Lemma 1.2: (a) in the ring for $c = 1$; (b) in the path for $C = 2$.

Proof: The problem is in Apx by Lemma 1.1. To prove the Apx-hardness, we consider the family of request graphs $\mathcal{R}$ defined as follows.

Mimic the proof of Theorem 1.1 replacing the gadget of Figure 1.2 with the gadget of Figure 1.5(a). With slight abuse of notation, the edge corresponding to an element x is also denoted x. It is easy to check that the same proof carries over to these new gadgets, and therefore the problem of finding the maximum number of edge-disjoint triangles in this class $\mathcal{R}$ of graphs is Apx-hard. Note that all the graphs built in this way are also tripartite, as shown in Figure 1.5(a).

Håstad proved [141] that Maximum Bounded Covering by 3-Sets is Apx-hard even restricted to instances for which we know that there exists a collection of mutually disjoint 3-subsets covering *all* the elements in the set. Therefore, we can assume without loss of generality that any optimal solution of MECT-B in a graph $R \in \mathcal{R}$ corresponds to a collection of mutually disjoint 3-subsets covering *all* the elements in the set. Hence, such an optimal solution of MECT-B restricted to each gadget G_i corresponding to the set $c_i = \{x, y, z\}$ satisfies:

(i) either it contains the three edges x, y, z corresponding to the elements in the set $c_i = \{x, y, z\}$; or

(ii) it contains *none* of the edges x, y, z.

Thinking of the graphs $R \in \mathcal{R}$ as instances of Ring Traffic Grooming, the key observation is that:

- in case *(i)*, the gadget G_i can be partitioned into 9 K_3's and 4 P_4's (see Figure 1.5(b));
- in case *(ii)*, the gadget $G_i - \{x, y, z\}$ can be partitioned into 8 K_3's and 4 P_4's (see Figure 1.5(c)).

It is easy to see that such a partition uses the maximum number of edge-disjoint triangles in the tripartite graph R, and only K_3's and P_4's are involved. By Lemma 1.2, this partition is an optimal solution of Ring Traffic Grooming for $C = 1$ in R. Let OPT be the number of vertices of such an optimal solution in R, and let t^* be the number of triangles

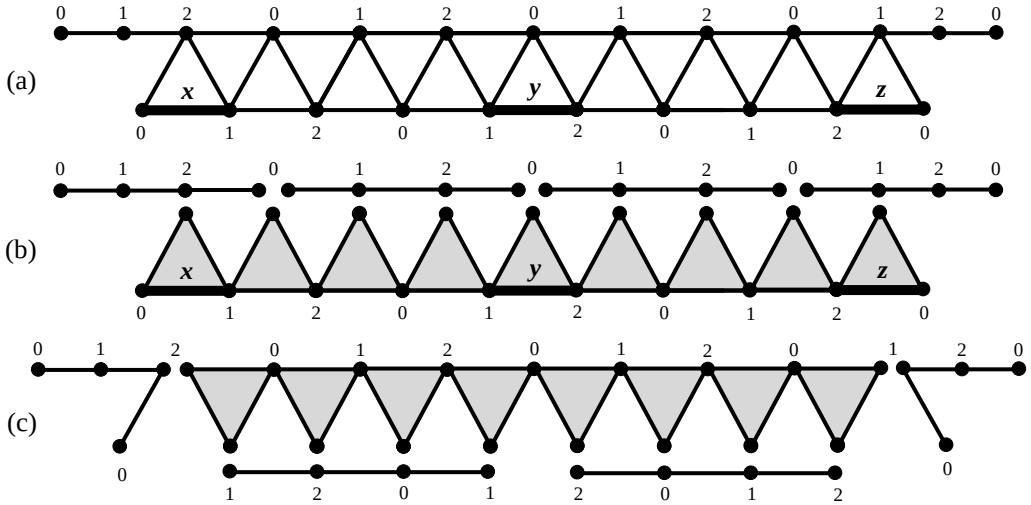

Figure 1.5: Request graphs used in the proof of Theorem 1.3: (a) gadget G_i corresponding to the set $c_i = \{x, y, z\}$. The labels of the vertices indicate the tripartition; (b) partition into 9 K_3's and 4 P_4's *with* the edges x, y, z ; (c) partition into 8 K_3's and 4 P_4's *without* the edges x, y, z.

in an optimal solution in R. (We simply write OPT and t^* instead of $OPT(R)$ and $t^*(R)$, respectively.) It is clear that

$$|E(R)| \leq OPT \leq 2|E(R)|. \tag{1.2}$$

We have seen in the proof of Lemma 1.2 that the cost A_t of any solution using t triangles satisfies $A_t \geq \frac{4}{3}|E(R)| - 3t$. We can also write

$$OPT = \frac{4}{3}|E(R)| - 3t^*. \tag{1.3}$$

Since MECT-B is Apx-hard in $\mathcal{R}$, there exists a constant $\varepsilon_0 > 0$ such that, unless P = NP, one cannot find in polynomial time more than $(1 - \varepsilon_0)t^*$ triangles in an arbitrary graph $R \in \mathcal{R}$. Therefore, the cost A of any solution of Ring Traffic Grooming that can be found in polynomial time satisfies

$$A \geq \frac{4}{3}|E(R)| - 3(1 - \varepsilon_0)t^* = OPT + 3\varepsilon_0 t^*, \tag{1.4}$$

where we have used Equation (1.3). On the other hand, from Equation (1.3) and using Equation (1.2) twice we get

$$t^* = \frac{4}{9}|E(R)| - \frac{OPT}{3} \geq \frac{4}{9}|E(R)| - \frac{|E(R)|}{3} = \frac{|E(R)|}{9} \geq \frac{OPT}{18}. \tag{1.5}$$

Combining Equations (1.4) and (1.5) yields that the cost A of any solution satisfies

$$A \geq OPT + 3\varepsilon_0 \frac{OPT}{18} = \left(1 + \frac{\varepsilon_0}{6}\right) OPT = (1 + \varepsilon_1)OPT,$$

with $\varepsilon_1 = \varepsilon_0/6 > 0$. Therefore, unless P = NP, Ring Traffic Grooming does not accept a PTAS for fixed $C = 1$. □

As expected, the result can be generalized to any $C \geq 1$.

Theorem 1.4 Ring Traffic Grooming *is* Apx*-complete for any fixed* $C \geq 1$*, even if the request graph has degree bounded by a constant* $B \geq 10$*. Thus, it does not accept a* PTAS *unless* P = NP*.*

Proof: The case $C = 1$ has been proved in Theorem 1.3, so assume henceforth that $C > 1$. The problem is in Apx for any $C \geq 1$ by Lemma 1.1. To prove the Apx-hardness, consider a $(2C + 1)$-partite graph as request graph, in such way that each cycle makes at least C tours around the center of the ring. At this point we can reduce the grooming problem to the problem of finding a maximum number of cycles of length $2C + 1$ in this graph (as in the case $C = 1$). This later problem is also Apx-complete by Theorem 1.2. The details follow.

Let G be a graph satisfying the conditions of Theorem 1.2: G is a $(2C + 1)$-partite graph, consisting of $2C + 1$ parts $A_0, \ldots, A_{2C}$ such that the only edges are between A_i and $A_{i+1 \pmod{2C+1}}$, $i = 0, \ldots, 2C$, and such that the graph induced between two consecutive parts of G forms a graph of girth at least $C + 1$. In order to simplify the presentation, suppose that this graph can be partitioned into C_{2C+1}'s.

Let $c_0, \ldots, c_{2C}$ be a permutation of the vertices of the cycle C_{2C+1}, such that the polygon $(c_0, \ldots, c_{2C})$ makes C tours around the center (for $C = 1$ take the triangle; for C arbitrary, let $c_i = \exp(\frac{2iC\pi}{2C+1})$). Now replace each vertex c_i with an interval consisting of vertices of A_i. In this cyclic representation of the graph G, each cycle makes at least C tours around the origin. To see this, recall that the only possible edges are between A_i and $A_{i+1 \pmod{2C+1}}$, $i = 0, \ldots, 2C$, and also that the graph induced between two consecutive parts has girth at least $C + 1$. This implies that every cycle should intersect each A_i at least once, and so this cycle makes at least C tours around the origin, as the original cycle $\{c_0, \ldots, c_{2C}\}$ does so.

Each cycle used in the solution should be of length exactly $2C + 1$, there is no cycle of smaller length, and longer cycles use each edge more than C times, as they make more than C tours around the origin. Then the problem is reduced to finding edge-disjoint cycles of length $2C + 1$, which is Apx-hard by Theorem 1.2. The proof of Theorem 1.3 can now be reproduced to obtain the same result for any C, replacing the factor $\frac{4}{3}$ for $C = 1$ (because the path with greatest density in any solution for $C = 1$ is a P_4) with a factor $\frac{2C+2}{2C+1}$ for a general C (because the path with greatest density in any solution for general C is P_{2C+2}). Hence, Ring Traffic Grooming is Apx-complete even for bounded number of requests per node $B \geq 10$. □

These ideas can be naturally extended to prove the Apx-completeness of Path Traffic Grooming for any fixed $C \geq 2$.

Theorem 1.5 Path Traffic Grooming *is* Apx*-complete for any fixed* $C \geq 2$*. Thus, it does not accept a* PTAS *unless* P = NP*.*

Proof: Again, the result holds even for bounded number B of requests per node, $B \geq 10$. We prove the result for $C = 2$, proceeding for $C > 2$ as in the proof of Theorem 1.4. Consider the family of request graphs $\mathcal{R}$ defined in the proof of Theorem 1.3, and place the three partition classes consecutively on the path one after another, as shown in Figure 1.4(b). Since each triangle induces load 2, minimizing the number of ADMs corresponds to finding the maximum number of edge-disjoint triangles. Therefore, the problem does not accept a PTAS unless P = NP. □

1.4 Approximating Traffic Grooming

We are now interested in finding good approximation algorithms considering C as part of the input. As discussed in Section 1.3, obtaining a $O(\sqrt{C})$-approximation is trivial. Since in practical applications SONET WDM rings are widely used as backbone optical networks [104, 162], the grooming factor is usually greater than the size of the network, i.e., $C \geq n$. Thus, it turns out to be important to find approximation algorithms with an approximation ratio not depending on C. A general approximation algorithm with this property is the main result of this section. It provides in the worst case a $O(n^{1/3} \log^2 n)$-approximation. We describe it for the ring topology, but exactly the same arguments provide an algorithm for the path. The main idea is to greedily find subgraphs with high density using approximation algorithms for the Dense k-Subgraph problem, which is defined as follows: given a graph G and an integer k, find an induced subgraph $H \subseteq G$ on k vertices with the greatest density among all subgraphs on k vertices. In [112] the authors provide a polynomial-time algorithm with approximation ratio $2n^{1/3}$. To simplify the presentation, suppose that $n = 2^t$ for some $t > 0$ (otherwise, introduce dummy vertices on the ring until getting size $n' = 2^t$, with $n' < 2n$):

Algorithm $\mathcal{A}$:

(1) Divide the request set into $\log n$ classes, such that in each class C_i the length of the requests lies in the interval $[2^i, 2^{i+1})$, $i = 0, \ldots, \log n - 1$. For each class C_i, the ring can be divided into intervals of length 2^i such that the only requests are between consecutive intervals. In this way we obtain $\frac{n}{2^i}$ subproblems for each class: each one consists in finding an optimal solution in a bipartite graph of size $2 \cdot 2^i$. More precisely, each subproblem can be formulated as:

Bipartite Traffic Grooming
Input: A bipartite graph R, and a grooming factor C.
Output: Partition of the edges of R into subgraphs R_λ such that $\lvert E(R_\lambda)\rvert \leq C$, for $1 \leq \lambda \leq \Lambda$.
Objective: Minimize $\sum_{\lambda=1}^{\Lambda} \lvert V(R_\lambda)\rvert$.

Find a solution to each Bipartite Traffic Grooming subproblem independently using step **(2)**, and output the union of all solutions.

(2) To find a solution to each Bipartite Traffic Grooming subproblem in a bipartite graph R, proceed greedily (until all edges are covered) by finding at step j a subgraph R_j of $G \setminus (R_1 \cup \cdots \cup R_{j-1})$ with at most C edges in the following way:
For each $k = 2, \ldots, 2C$ find a subgraph B_k of $R \setminus (R_1 \cup \cdots \cup R_{j-1})$ using the algorithm of [112] for the Dense k-Subgraph problem.
- If for some k^*, $|E(B_{k^*})| > C$, and $|E(B_j)| \leq C$ for all $j < k^*$, remove $|E(B_{k^*})| - C$ arbitrary edges from B_{k^*} and output the densest graph among $B_2, \ldots, B_{k^*-1}, B_{k^*}$.
- Otherwise, output the densest graph among $B_2, \ldots, B_{2C}$.

Let OPT be an optimal solution of Ring Traffic Grooming, and let OPT_1 be the cost of the solution obtained by solving *optimally* all the subproblems generated by step **(1)** of Algorithm $\mathcal{A}$. We prove a lemma before stating Theorem 1.6.

Lemma 1.3 *Let β be a given positive real number. Suppose that there exists an algorithm that finds in any bipartite graph R on at most n vertices, a subgraph with at most C edges which has density at least $1/\beta$ times the density of the densest subgraph with at most C edges. Then in the greedy procedure of step **(2)** of Algorithm $\mathcal{A}$, one obtains a solution of cost OPT_2 such that $OPT_2 \leq O(\log n) \cdot \beta \cdot OPT_1$.*

Proof: Let m be the number of edges of the request graph R, and let $R_1, R_2, \ldots, R_r$ be the subgraphs generated, in this order, by the above algorithm. We will prove that $\sum |V(R_i)| \leq \log(m) \cdot \beta \cdot OPT_1$. To prove this, we first enumerate the edges of R in order of appearance in R_i's: all the edges in R_1 will be enumerated $e_1, \ldots, e_{C_1}$ ($C_1 = |E(R_1)| \leq C$), all the edges in R_2 will be enumerated $e_{C_1+1}, \ldots, e_{C_1+C_2}$ ($C_2 = |E(R_2)| \leq C$), and so on. Let ρ_i be the density of the subgraph R_i, i.e., $\rho_i = \frac{|E(R_i)|}{|V(R_i)|}$, and $\Sigma = \sum |V(R_i)|$ the total cost of the solution. For every edge $e_j \in R_i$, we define $c(e_j) = \frac{1}{\rho_i}$. We claim that $\sum_j c(e_j) = \Sigma$. To prove this equality just note that $\sum_{e_j \in E(R_i)} c(e_j) = \frac{|E(R_i)|}{\rho_i} = |V(R_i)|$, and so $\sum_j c(e_j) = \sum_i |V(R_i)| = \Sigma$. Let us define R'_i to be the union of $R_i, R_{i+1}, \ldots, R_r$. We define ρ'_i to be the density of the densest subgraph of R'_i containing at most C edges. Let us take an optimal solution for R'_i, i.e., a decomposition of R'_i into subgraphs $A_1, \ldots, A_s$ such that $\sum_{k=1}^{s} |V(A_k)|$ is minimum. Let $\overline{\rho}_1, \ldots, \overline{\rho}_s$ be the density of these subgraphs. We have:

- $\forall k \leq s$, $\overline{\rho}_k \leq \rho'_i$: because each A_k is a subgraph of R'_i containing at most C edges, and ρ'_i is the density of the densest subgraph with at most C edges in R'_i.
- $\rho'_i \leq \beta\rho_i$: because we suppose that we can find an approximation of ρ'_i up to a factor $1/\beta$.

This implies that $\frac{1}{\overline{\rho}_k} \geq \frac{1}{\beta\rho_i}$, and so

$$\sum_k |V(A_k)| = \sum_k \frac{|E(A_k)|}{\overline{\rho}_k} \geq \sum_k \frac{|E(A_k)|}{\beta\rho_i} = \frac{|E(R'_i)|}{\beta\rho_i}.$$

But an optimal solution for R provides a solution for R'_i of cost at least the optimal solution for R'_i, i.e., $\sum_k |V(A_k)| \leq OPT_1$. Using this in the above inequality we get $\frac{1}{\rho_i} \leq \frac{\beta \cdot OPT_1}{|E(R'_i)|}$, and so for an edge $e_j \in R_i$ we have $c(e_j) = \frac{1}{\rho_i} \leq \frac{\beta \cdot OPT_1}{|E(R'_i)|} \leq \frac{\beta \cdot OPT_1}{m-j+1}$, and this proves that

$$\Sigma = \sum_j c(e_j) \leq \beta \cdot (\sum_j \frac{1}{m-j+1}) \cdot OPT_1 \leq \beta \cdot \log(m) \cdot OPT_1 \leq 2\beta \cdot \log(n) \cdot OPT_1.$$

□

Theorem 1.6 *$\mathcal{A}$ is a poly-time approximation algorithm that approximates* Ring Traffic Grooming *within a factor $O(n^{1/3} \log^2 n)$ for any $C \geq 1$.*

Proof: Algorithm $\mathcal{A}$ returns a valid solution of Ring Traffic Grooming, because each request is contained in some bipartite graph, and no request is counted twice. The runtime is polynomial in both n and C, because we run at most $2C-1$ times the algorithm of [112] for each subproblem, and there are $n(\sum_{i=0}^{t-1} \frac{1}{2^i}) - 1 = 2n - 3$ subproblems. We prove the approximation guarantee:

- We claim that $OPT_1 \leq 2 \log n \cdot OPT$. Indeed, let $\overline{c}_i$ be the optimal cost of the subset of requests of length in the interval $[2^i, 2^{i+1})$, $i = 0, \ldots, \log(n) - 1$. It is clear that $\overline{c}_i \leq OPT$ for each i , and thus $\sum_{i=0}^{\log n - 1} \overline{c}_i \leq \log n \cdot OPT$. Finally, $OPT_1 \leq 2 \sum_{i=0}^{\log n - 1} \overline{c}_i$, because each vertex is taken into account in two subproblems.

- The greedy procedure described in step **(2)** of Algorithm $\mathcal{A}$ outputs a graph whose density is at least $\frac{1}{2n^{1/3}}$ times the greatest density (with at most C edges) of the updated request graph. To see that, note that the optimal density is achieved by a subgraph on at most $2C$ vertices (it would be the case of C disjoint edges). Then, for each value of k, the algorithm of [112] finds a $2n^{1/3}$-approximation of the maximum number of edges of an induced subgraph on k vertices[1]. Thus, if we take the densest subgraph among $B_2, \ldots, B_{2C}$ (removing edges if necessary) we also obtain a $2n^{1/3}$-approximation of the greatest density of a subgraph with at most C edges. Let ρ_k be the density of B_k before removing edges. The explicit formula of the greatest density ρ that we output in step **(2)** of Algorithm $\mathcal{A}$ is:

  $$\rho = \max_{k \in \{2, \ldots, 2C\}} \min\left(\rho_k, \frac{C}{k}\right).$$

  The above formula justifies that the algorithm stops the search at $k = k^*$. Summarizing, we can use $\beta = 2n^{1/3}$ in Lemma 1.3.

- By combining the remarks above and Lemma 1.3 we obtain that the cost A returned by Algorithm $\mathcal{A}$ satisfies $A \leq 2n^{1/3} \cdot OPT_2 \leq 4n^{1/3} \log n \cdot OPT_1 \leq 8n^{1/3} \log^2 n \cdot OPT$.

□

[1] In fact, the improved approximation ratio of the Dense k-Subgraph problem is $O(n^\delta)$ for some constant $\delta < 1/3$ [112]. Obviously, the same applies to our algorithm, replacing the exponent 1/3 with the same $\delta < 1/3$.

We can improve the approximation ratio of the algorithm if all the requests have short length compared to the length of the ring. This situation is usual in practical applications since nodes may want to communicate only with their nearest neighbors. Let $f(n)$ be any function of n. If all the requests have length at most $f(n)$, then the above algorithm provides an approximation ratio of $O(f(n)^{1/3}\log^2 n)$. Indeed, in step **(2)** of Algorithm $\mathcal{A}$, we have to find dense subgraphs in bipartite graphs of size at most $2f(n)$, hence the factor $2n^{1/3}$ can be replaced with $2(2f(n))^{1/3}$.

Remark that all the instances of Dense k-Subgraph problem in our algorithm are bipartite. Using the results of [187], it is possible to obtain a better approximation ratio when the request graph is bipartite and satisfies some uniform density conditions.

Corollary 1.1 *If the request graph R is such that in any large enough subgraph $H \subseteq R$, a densest subgraph $(A \cup B, E)$ satisfies $|A|, |B| = O(\sqrt{C})$ and $|E| = \Omega(C)$, then for any constant $\varepsilon > 0$ there exists a polynomial-time algorithm for* Ring Traffic Grooming *with approximation ratio $O(n^{\varepsilon}\log^2 n)$.*

To end this section, it is interesting to mention that the results of [90] show that the density can be approximated within a constant factor two in the class of graphs excluding a fixed graph H as minor. Thus, if the request graph R is H-minor free (for instance if R is planar or of bounded genus), Algorithm $\mathcal{A}$ achieves an approximation factor of $O(\log^2 n)$.

1.5 Conclusions

The contribution of this chapter can be divided into two main parts: on the one hand, we stated inapproximability results for Ring Traffic Grooming and Path Traffic Grooming for fixed values of C. More precisely, we proved that Ring Traffic Grooming is Apx-complete for fixed $C \geq 1$, and that Path Traffic Grooming is Apx-complete for fixed $C \geq 2$. In other words, we ruled out the existence of a PTAS for fixed values of C. To prove this results we reduced Ring Traffic Grooming for $C = 1$ to the problem of finding the maximum number of edge-disjoint triangles in a graph of degree bounded by B (MECT-B for short). We proved that MECT-B is Apx-complete, and we generalized this reduction for Path Traffic Grooming and for all values of $C \geq 1$. On the other hand, we provided a polynomial-time approximation algorithm for Ring and Path Traffic Grooming with an approximation ratio not depending on C, considering C as part of the input.

A number of interesting questions remain open. First, when C is not part of the input, the non-existence of a PTAS blows the whistle to start the race of finding the best constant factor approximation for each value of C, for both the ring ($C \geq 1$) and the path ($C \geq 2$). We did not focus on this issue in this chapter.

Secondly, when C is part of the input, it is a challenging open problem to close the complexity gap of Traffic Grooming, that is, to provide an approximation algorithm with an approximation ratio matching the corresponding inapproximability result. We are convinced that the inherent difficulty of the problem resides in finding dense subgraphs with bounded number of edges. This problem is strongly related to the problem of finding

the densest subgraph with bounded number of vertices, which has been recently proved to have, essentially, the same difficulty as the DENSE k-SUBGRAPH problem [36]. The non-existence of a PTAS for the DENSE k-SUBGRAPH problem has been proved in [152] involving very technical proofs, and this is the best existing hardness result. A long-standing conjecture claims that there exists some constant $\varepsilon > 0$ such that finding a n^{ε}-approximation for DENSE k-SUBGRAPH is NP-hard [112]. As we proved in Section 1.4, an α-approximation for DENSE k-SUBGRAPH yields a $O(\alpha \log^2 n)$-approximation for RING TRAFFIC GROOMING. We suspect that a similar result in the other direction should also exist. Because of this, we conjecture that:

Conjecture 1.1 *There exists some constant $\delta > 0$, such that* RING TRAFFIC GROOMING *is* NP*-hard to approximate in polynomial time within a factor $O(n^{\delta})$ when the grooming factor C is part of the input.*

In [72, Proposition 2] it is shown that RING TRAFFIC GROOMING is in P for fixed n. Using terminology from parameterized complexity (see Section I.2.3), this result only shows that RING TRAFFIC GROOMING is in XP and not necessarily FPT (if n is the parameter). In [72] it is said that Michael Fellows has shown that if the number of ADMs is taken to be the parameter, then RING TRAFFIC GROOMING is in FPT. Unfortunately, the number of ADMs tends to be much larger than the ring size, so it remains an interesting open problem whether ring grooming is FPT if n is the parameter and C is part of the input.

Chapter 2

Bounded-degree Request Graph

In this chapter we consider the unidirectional ring topology with a generic grooming factor C. We introduce the pseudo-dynamic case where the request graph has bounded degree Δ, and our aim is to design a network (namely, place the ADMs at each node) being able to support *any* request graph with maximum degree at most Δ. The existing theoretical models in the literature are much more rigid, and do not allow such adaptability. We show that the problem is essentially equivalent to finding the least integer $M(C, \Delta)$ such that the edges of any graph with maximum degree at most Δ can be partitioned into subgraphs with at most C edges and each vertex appears in at most $M(C, \Delta)$ subgraphs. We establish the value of $M(C, \Delta)$ for almost all values of C and Δ, leaving open only the case where $\Delta \geq 5$ is odd, $\Delta \pmod{2C}$ is between 3 and $C - 1$, $C \geq 4$, and the request graph does not contain a perfect matching. For these open cases, we provide upper bounds that differ from the optimal value by at most one.

Keywords: optical networks, SONET over WDM, traffic grooming, ADM, graph decomposition, cubic graph, perfect matching, bridgeless graph.

2.1 Introduction

In this chapter we consider unidirectional SONET/WDM ring networks with symmetric requests. As already mentioned in Section II.2 (page 25), in this case the routing is unique and to each request between two nodes, we assign a wavelength and some bandwidth on this wavelength. If the traffic is uniform and any given wavelength can carry at most C requests, we can assign at most $\frac{1}{C}$ of the bandwidth to each request, C being the grooming factor. Furthermore, if the traffic requirement is symmetric, we may assume that symmetric requests are assigned the same wavelength, as it is easy to show (by exchanging wavelengths) that there exists an optimal solution where all symmetric requests are given

the same wavelength. Then each pair of symmetric requests uses $\frac{1}{C}$ of the bandwidth in the whole ring. If the two end-nodes are u and v, we need one ADM at node u and one at node v. The main point is that if two requests have a common end-node, they can share an ADM if they are assigned the same wavelength.

The traffic grooming problem for a unidirectional SONET ring with n nodes, grooming ratio C, and a symmetric request graph R has been modeled as a graph partition problem as follows (see [47,136]). Each edge of R corresponds to a pair of symmetric requests, and edges are colored by their assigned wavelength λ. All edges of color λ induce a connected subgraph B_λ of R, where each node corresponds to an ADM. The grooming constraint, i.e., the fact that a wavelength can carry at most C requests, translates to an upper bound C on the number of edges in each B_λ. The cost corresponds to the total number of vertices used in the subgraphs, and the objective is therefore to minimize $\sum_\lambda |V(B_\lambda)|$.

While most of previous work has focused on the case where the requests are given as input [J2,47,48,104,116,118,136,162], we consider the case where only the network topology is given, together with a bound Δ on the request graph. We would like to place, for each value of the grooming factor C, a minimum number of ADMs at each node in such a way that they could support *any* traffic pattern where each node is the end-node of at most Δ requests. This model is interesting because the network can support dynamic traffic without replacement of the ADMs.

From a practical point of view, it is interesting to design a network being able to support any request graph with maximum degree not exceeding a given constant. This situation is usual in real optical networks, since due to technology constraints the number of allowed communications for each node is usually bounded. This flexibility can also be thought from another point of view: if we have a limited number of available ADMs to place at the nodes of the network, then it is interesting to know which is the maximum degree of a request graph that our network is able to support, depending on the grooming factor. Equivalently, given a maximum degree and a number of available ADMs, it is useful to know which values of the grooming factor the network will support.

The aim of this chapter is to provide a theoretical framework to design such networks with dynamically changing traffic. We study the case where the physical network is given by an unidirectional ring, which is a widely used topology (for instance, in SONET rings). The problem we study can be formulated as a graph partitioning problem as follows.

Δ-Degree-Bounded Traffic Grooming in Unidirectional Rings
Input: Three integers n (size of the ring), C (grooming factor), and Δ (maximum degree).
Output: An assignment of $A(v)$ ADMs to each vertex v of the ring, in such a way that *for any request graph* G with maximum degree at most Δ, there exists a partition of $E(G)$ into subgraphs $\{B_\lambda\}_{1\leq\lambda\leq\Lambda} = \mathcal{B}$, such that:
(i) $|E(B_\lambda)| \leq C$ for all λ; and
(ii) each vertex $v \in V(G)$ appears in at most $A(v)$ subgraphs.
Objective: Minimize $\sum_v A(v)$.

The optimum to the above problem for each n, C, Δ is denoted by $A(n, C, \Delta)$. Before getting into details, we first fix the notation to be used in this chapter.

Notation. The (multi)graphs considered in this chapter are finite and without self-loops. A Δ-*graph* is a (multi)graph with maximum degree at most Δ. $\mathcal{G}_\Delta$ denotes the class of all Δ-graphs. A Δ-*regular* (multi)graph is a graph in which all vertices have degree Δ. An *almost* Δ-*regular* (multi)graph is a (multi)graph in which all vertices have degree Δ except possibly one which has degree $\Delta-1$. A *bridge* in a (multi)graph G is an edge whose removal disconnects G. A *matching* in a (multi)graph $G = (V, E)$ is a subset $M \subseteq E$ which contains each vertex at most once. A *perfect matching* is a matching containing all vertices. A *digon* is a cycle of length 2. A *trail* in a (multi)graph is a sequence $\{\{x_1, x_2\}, \{x_2, x_3\}, \ldots, \{x_{k-1}, x_k\}\}$ of distinct edges in which the second end of an edge is the first end of the next edge (the same pair of vertices may appear more than once if there is more than one edge between them). Vertices $x_2, x_3, \ldots, x_{k-1}$ of a trail are called *midpoints*. The *length* of a trail is the number of edges in it. Given a (multi)graph $G = (V, E)$ and a subset of vertices $V' \subseteq V$, we denote by $G - V'$ the (multi)graph obtained from G by removing the vertices in V', the edges incident with vertices in V', and isolated vertices (if any). Similarly, given a subset of edges $E' \subseteq E$, we denote by $G - E'$ the (multi)graph obtained from G by removing the edges in E' and isolated vertices (if any). Given a graph with maximum degree at most Δ, a partition of G into subgraphs with at most C edges is called a C-*edge-partition* of G.

The function $A(n, C, \Delta)$ satisfies some straightforward properties.

Lemma 2.1 *The following statements hold:*

(i) $A(n, C, 1) = n$.

(ii) $A(n, 1, \Delta) = \Delta \cdot n$.

(iii) *If* $C' \geq C$, *then* $A(n, C', \Delta) \leq A(n, C, \Delta)$.

(iv) *If* $\Delta' \geq \Delta$, *then* $A(n, C, \Delta') \geq A(n, C, \Delta)$.

(v) $A(n, C, \Delta) \geq n$ *for all* $\Delta \geq 1$.

(vi) *If* $C \geq \frac{n\Delta}{2}$, $A(n, C, \Delta) = n$.

Proof:

(i) The request graph can consist in a perfect matching, so any solution uses 1 ADM per node.

(ii) A Δ-regular graph can be partitioned into $\frac{n\Delta}{2}$ disjoint edges.

(iii) Any solution for C is also a solution for C'.

(iv) If $\Delta' \geq \Delta$, the subgraphs with maximum degree at most Δ are a subclass of the class of graphs with maximum degree at most Δ'.

(v) Combine *(i)* and *(iv)*.

(vi) In this case all the edges of the request graph fit into one subgraph.

□

Organization of the chapter. In Section 2.2 we show that the Δ-Degree-Bounded Traffic Grooming in Unidirectional Rings problem is essentially equivalent to establishing the value of the parameter $M(C, \Delta)$ (see Definition 2.1) for each value of C and Δ. We solve the cases where $\Delta \geq 2$ is even in Section 2.3. In Section 2.4 we focus on the cases where $\Delta \geq 3$ is odd, leaving open only the cases where $\Delta \geq 5$ is odd, $\Delta \pmod{2C}$ is between 3 and $C-1$, $C \geq 4$, and the graph does not contain a perfect matching (see Table 2.1). In Section 2.4.6 we present an attempt to solve these remaining cases, that may lead to an eventual proof. Finally, Section 2.5 concludes the chapter.

2.2 The Parameter $M(C, \Delta)$

The following definition will play a fundamental role in the remainder of this chapter.

Definition 2.1 *Let $M(C, \Delta)$ be the smallest number M such that $A(n, C, \Delta) \leq M \cdot n$ for all $n \geq 1$.*

Lemma 2.2 *$M(C, \Delta)$ is a natural number.*

Proof: We know by Lemma 2.1 that, for any $C \geq 1$, $n \leq A(n, C, \Delta) \leq A(n, 1, \Delta) = \Delta \cdot n$. Suppose that M is not a natural number. That is, suppose that $r < M < r+1$ for some positive integer r. Therefore, there must be at least $(r+1-M)n$ vertices with at most r ADMs each. For each n, let $V_{n,r}$ be the subset of vertices of the request graph with at most r ADMs. Then, since $r+1-M > 0$, we have that $\lim_{n\to\infty} |V_{n,r}| = \infty$. In other words, there is an arbitrarily big subset of vertices with at most r ADMs per vertex. But we can consider a request graph with maximum degree at most Δ on the set of vertices $V_{n,r}$, and this means that with r ADMs per node we can construct a C-edge-partition, a contradiction with the optimality of M. □

If the request graph is further restricted to belong to a subclass of graphs $\mathcal{C} \subseteq \mathcal{G}_\Delta$, then the corresponding positive integer is denoted by $M(C, \Delta, \mathcal{C})$.

By the discussion above, $A(n, C, \Delta)$ is of the form $A(n, C, \Delta) = M(C, \Delta) \cdot n - \alpha(C, \Delta)$, where $M(C, \Delta)$ and $\alpha(C, \Delta)$ are integers depending only on C and Δ. Suppose that a Δ-graph H requires at least $M(C, \Delta) + 1$ ADMs at some vertex. Since any Δ-graph must be supported with the same ADMs, by relabeling the vertices of H we could force at least $M(C, \Delta) + 1$ ADMs in $\Omega(n)$ nodes of the network. This would contradict the definition of $M(C, \Delta)$. Therefore, each vertex can appear in at most $M(C, \Delta)$ subgraphs. So we may conclude the following.

Remark 2.1 *For each value of C and Δ, Δ-Degree-Bounded Traffic Grooming in Unidirectional Rings reduces to finding the least integer $M(C, \Delta)$ such that the edges of any Δ-graph can be partitioned into subgraphs with at most C edges and each vertex appears in at most $M(C, \Delta)$ subgraphs.*

This allows us to give an equivalent definition of $M(C,\Delta)$. Let $G \in \mathcal{G}_\Delta$ and let $\mathcal{P}_C(G)$ be the set of C-edge-partitions of G. For $P \in \mathcal{P}_C(G)$, let $\mathrm{occ}(P)$ be the maximum number of occurrences of a vertex in the partition, that is,

$$\mathrm{occ}(P) = \max_{v \in V(G)} |\{B_\lambda \in P : v \in B_\lambda\}|, \quad \text{and then} \quad M(C,\Delta) = \max_{G \in \mathcal{G}_\Delta} \left(\min_{P \in \mathcal{P}_C(G)} \mathrm{occ}(P) \right).$$

In the remainder of this chapter, we use Remark 2.1 and focus on determining $M(C,\Delta)$ for each value of C and Δ. Observe also that any Δ-graph H is a subgraph of some Δ-regular graph G (with possibly more vertices). Note also that if we restrict a partition of G to the vertices of H, the number of occurrences of the vertices cannot increase. Therefore,

Remark 2.2 *$M(C,\Delta) = M(C,\Delta,\mathcal{C})$, where $\mathcal{C}$ is the class of Δ-regular graphs.*

The following lemma will be used throughout the chapter.

Lemma 2.3 *The following statements hold trivially from Lemma 2.1:*

(i) $M(C,1) = 1$ *for all* $C \geq 1$.

(ii) $M(1,\Delta) = \Delta$ *for all* $\Delta \geq 1$.

(iii) *If* $C' \geq C$, *then* $M(C',\Delta) \leq M(C,\Delta)$.

(iv) *If* $\Delta' \geq \Delta$, *then* $M(C,\Delta') \geq M(C,\Delta)$.

(v) $M(C,\Delta) \leq \Delta$ *for all* $C,\Delta \geq 1$.

The following proposition establishes a general lower bound on $M(C,\Delta)$, that will allow us to prove in many cases the optimality of the constructions of the next sections.

Proposition 2.1 $M(C,\Delta) \geq \left\lceil \frac{C+1}{C} \frac{\Delta}{2} \right\rceil$ *for all values of* C,Δ.

Proof: Erdös and Sachs [111] proved that for any integer k there exist k-regular graphs with arbitrary large girth. For each value of C and Δ, let G be a Δ-regular graph with girth at least $C+1$, and let $n = |V(G)|$. Clearly all the subgraphs (with at most C edges) involved of the partition of the $\Delta n/2$ edges of G are trees. Therefore, the total number of vertices of any partition is at least $\frac{\Delta(C+1)}{2C} n$ (this can be easily seen using that the function $(x+1)/x$ with $1 \leq x \leq C$ is minimized when $x = C$). Then necessarily a vertex must occur in at least $\frac{\Delta(C+1)}{2C}$ subgraphs. By the definition of $M(C,\Delta)$, the lower bound follows. □

2.3 Case $\Delta \geq 2$ Even

In this section we establish the value of $M(C, \Delta)$ for $\Delta \geq 2$ even and any value of C.

Theorem 2.1 *Let $\Delta \geq 2$ be even. Then for any $C \geq 1$, $M(C, \Delta) = \left\lceil \frac{C+1}{C} \frac{\Delta}{2} \right\rceil$.*

Proof: The lower bound follows from Proposition 2.1. Let us give an explicit construction for any Δ-regular graph $G = (V, E)$. Orient the edges of G in an Eulerian tour, and assign to each vertex $v \in V$ its $\Delta/2$ out-edges, namely E_v^+. For each $v \in V$, partition E_v^+ into $\left\lceil \frac{\Delta}{2C} \right\rceil$ stars with C edges centered at v (except, possibly, one star with fewer edges). Each vertex v appears as a leaf in stars centered at other vertices exactly $\Delta - \Delta/2 = \Delta/2$ times. Therefore, the number of occurrences of each vertex in this partition is

$$\left\lceil \frac{\Delta}{2C} \right\rceil + \frac{\Delta}{2} = \left\lceil \frac{\Delta}{2} \left(1 + \frac{1}{C} \right) \right\rceil = \left\lceil \frac{C+1}{C} \frac{\Delta}{2} \right\rceil. \qquad \square$$

Note that for the special case $\Delta = 2$, Theorem 2.1 implies that $M(C, 2) = 2$ for all $C \geq 1$. In fact, for $\Delta = 2$ it is possible to give the exact expression of the cost function $A(n, C, 2)$. Indeed, it is easy to see that given a set of disjoint cycles, we can always find a C-edge-partition such that $C - 1$ prescribed (arbitrary) vertices appear in only one subgraph. On the other hand, if we pretend that at least C vertices appear in at most one subgraph, we can consider as request graph a cycle of length at least $C + 1$ containing the prescribed C vertices, and then necessarily one of those vertices appears in at least two subgraphs of any C-edge-partition, a contradiction. Summarizing, $A(n, C, 2) = 2n - (C - 1)$.

2.4 Case $\Delta \geq 3$ Odd

The cases where Δ is odd turn out to be inherently much more complicated than the cases where Δ is even. In Section 2.4.1, we present a general construction which differs from the lower bound of Proposition 2.1 by at most 1, and we determine when this construction is optimal. In Section 2.4.2 we provide an improved lower bound when $\Delta \equiv C \pmod{2C}$, which meets our upper bound. In Section 2.4.4 we solve the case $\Delta = 3$ and $C = 4$, which was the only unsolved case for $\Delta = 3$. In Section 2.4.5 we present an optimal construction for graphs with a perfect matching, after proving that the lower bound of Proposition 2.1 still holds when the request graph is restricted to have a perfect matching. We then discuss in Section 2.4.3 the relation of the parameter $M(C, \Delta)$ with the *linear C-arboricity* [33, 52, 190]. Finally, we describe in Section 2.4.6 an attempt to solve the remaining cases where $\Delta \geq 5$ is odd, using the ideas developed in the previous sections.

2.4.1 General upper bound

The following proposition provides a general upper bound, which differs from the lower bound of Proposition 2.1 by at most 1.

Proposition 2.2 *Let $\Delta \geq 3$ be odd. Then for any $C \geq 1$, $M(C, \Delta) \leq \left\lceil \frac{C+1}{C} \frac{\Delta}{2} + \frac{C-1}{2C} \right\rceil$.*

Proof: Let G be a Δ-regular graph. Since Δ is odd, $|V(G)|$ is even. Add a perfect matching M to G to obtain a $(\Delta+1)$-regular multigraph G'. Orient the edges of G' in an Eulerian tour, and assign to each vertex $v \in V(G')$ its $(\Delta+1)/2$ out-edges E_v^+. Remove the edges of M and, as in the case Δ even, partition E_v^+ into stars with at most C edges. To count the number of occurrences of each vertex, we distinguish two cases. If an edge of M is in E_v^+, then v appears as center in $\left\lceil \frac{\Delta-1}{2C} \right\rceil$ stars and as a leaf in $\Delta - \frac{\Delta-1}{2}$ stars. Summing both terms yields

$$\left\lceil \frac{\Delta-1}{2C} \right\rceil + \Delta - \frac{\Delta-1}{2} = \left\lceil \frac{C+1}{C}\frac{\Delta}{2} + \frac{C-1}{2C} \right\rceil.$$

Otherwise, if no edge of M is in E_v^+, the number of occurrences of v is

$$\left\lceil \frac{\Delta+1}{2C} \right\rceil + \Delta - \frac{\Delta+1}{2} = \left\lceil \frac{C+1}{C}\frac{\Delta}{2} + \frac{1-C}{2C} \right\rceil \leq \left\lceil \frac{C+1}{C}\frac{\Delta}{2} + \frac{C-1}{2C} \right\rceil.$$

□

The upper bound of Proposition 2.2 and the lower bound of Proposition 2.1 are equal for, roughly speaking, half of the pairs C, Δ, as shown in the following corollary.

Corollary 2.1 *Let $\Delta \geq 3$ be odd. If $\Delta \pmod{2C} = 1$ or $\Delta \pmod{2C} \geq C+1$, then $M(C,\Delta) = \left\lceil \frac{C+1}{C}\frac{\Delta}{2} \right\rceil$.*

Proof: Let $\Delta = \lambda \cdot 2C + h$, with h odd, $1 \leq h \leq 2C-1$. Writing $k := \lambda(C+1) + \frac{h-1}{2}$, the lower bound of Proposition 2.1 equals $k + \left\lceil \frac{1}{2} + \frac{h}{2C} \right\rceil$, and the upper bound of Proposition 2.2 equals $k + \left\lceil 1 + \frac{h-1}{2C} \right\rceil$. If $h = 1$ both bounds equal $k+1$, and if $h \geq C+1$ both bounds equal $k+2$. □

In particular, when $C = 2$ and Δ is odd, $\Delta \pmod{2C}$ is either 1 or 3, and then by Corollary 2.1 the lower bound is attained, as stated in the following corollary.

Corollary 2.2 (Case $C = 2$) *For any $\Delta \geq 3$ odd, $M(2,\Delta) = \left\lceil \frac{3\Delta}{4} \right\rceil$.*

For all the cases we solved so far, the value of $M(C,\Delta)$ equals the lower bound of Proposition 2.1. It seems natural to think that the value $\left\lceil \frac{C+1}{C}\frac{\Delta}{2} \right\rceil$ may be always attained. We shall see in the next section that this is not true. Namely, we prove in Theorem 2.2 that if $\Delta \equiv C \pmod{2C}$, then $M(C,\Delta) = \left\lceil \frac{C+1}{C}\frac{\Delta}{2} \right\rceil + 1$.

2.4.2 Improved lower bound

In this section we prove a new lower bound which strictly improves on Proposition 2.1 when $\Delta \equiv C \pmod{2C}$.

Theorem 2.2 *Let $\Delta \geq 3$ be odd and let $\Delta \equiv C \pmod{2C}$. Then $M(C,\Delta) = \left\lceil \frac{C+1}{C}\frac{\Delta}{2} \right\rceil + 1$.*

Proof: We prove that if $\Delta = kC$ with k odd, then $M(C,\Delta) \geq \left\lceil \frac{C+1}{C}\frac{\Delta}{2} \right\rceil + 1$ and thus, by Proposition 2.2, $M(C,\Delta)$ is equal to $\left\lceil \frac{C+1}{C}\frac{\Delta}{2} \right\rceil + 1$. Since both Δ and k are odd, so is C, and therefore $\left\lceil \frac{C+1}{C}\frac{\Delta}{2} \right\rceil = k \cdot \frac{C+1}{2}$.

We proceed to build a Δ-regular graph G with no C-edge-partition where each vertex is incident to at most $\left\lceil \frac{C+1}{C}\frac{\Delta}{2} \right\rceil$ subgraphs, hence implying that $M(C,\Delta) > \left\lceil \frac{C+1}{C}\frac{\Delta}{2} \right\rceil$. First, we construct a graph H where all vertices have degree Δ except one which has degree $\Delta - 1$. Furthermore, we build H so that it has girth strictly greater than C. H exists by [67,111]. Make Δ copies of H and add a cut-vertex v joined to all vertices of degree $\Delta - 1$ to make our Δ-regular graph G (see Figure 2.1 for an example of the construction of such a graph for $\Delta = C = 3$).

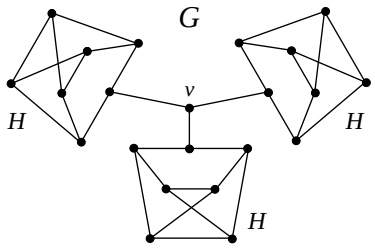

Figure 2.1: Cubic graph with girth 4, which is a counterexample showing that $M(3,3) = 3$.

Now suppose for the sake of contradiction that there is a C-edge-partition $\mathcal{B}$ of G where each vertex is incident to at most $\left\lceil \frac{C+1}{C}\frac{\Delta}{2} \right\rceil$ subgraphs. Since the girth of G is greater than C, all the subgraphs in $\mathcal{B}$ are trees. Since $\left\lceil \frac{C+1}{C}\frac{\Delta}{2} \right\rceil < \Delta$, v must have degree at least 2 in some subgraph $T' \in \mathcal{B}$. Since $|E(T')| \leq C$, the tree T' contains at most $\left\lfloor \frac{C-2}{2} \right\rfloor = \frac{C-3}{2}$ edges of a copy H' of H intersecting T'. Now we only work in H'. Let $\alpha = |E(T' \cap H')| \leq \frac{C-3}{2}$ (note that $\alpha = 0$ for $C = \Delta = 3$).

Let $\mathcal{B}' = \{B \cap H'\}_{B \in (\mathcal{B} - \{T'\})}$, with the empty subgraphs removed. That is, $\mathcal{B}'$ contains the subgraphs in $\mathcal{B}$ that partition the edges in H' that are not in T'. Let $n = |V(H')|$, which is odd as in H' there is one vertex of degree $\Delta - 1$ and all the others have degree Δ. Therefore, the total number of edges of the trees in $\mathcal{B}'$ is

$$\sum_{T \in \mathcal{B}'} |E(T)| \;=\; |E(H')| - \alpha \;=\; \frac{n\Delta - 1}{2} - \alpha \;=\; \frac{nkC - 1}{2} - \alpha. \tag{2.1}$$

As $\alpha \leq \frac{C-3}{2}$, from Equation (2.1) we get

$$\sum_{T \in \mathcal{B}'} |E(T)| \;\geq\; \frac{nkC - 1}{2} - \frac{C-3}{2} \;=\; \left(\frac{nk-1}{2}\right) \cdot C + 1. \tag{2.2}$$

As each tree in $\mathcal{B}'$ has at most C edges, from Equation (2.2) we get that $|\mathcal{B}'|$, the number of trees in $\mathcal{B}'$, satisfies

$$|\mathcal{B}'| \;\geq\; \left\lceil \frac{nk-1}{2} + \frac{1}{C} \right\rceil \;=\; \frac{nk-1}{2} + \left\lceil \frac{1}{C} \right\rceil \;=\; \frac{nk-1}{2} + 1. \tag{2.3}$$

Clearly, the total number of vertices in the trees in $\mathcal{B}'$ is exactly the total number of edges in the trees in $\mathcal{B}'$ plus the number of trees in B', that is, $\sum_{T\in\mathcal{B}'}|V(T)| = \sum_{T\in\mathcal{B}'}|E(T)| + |\mathcal{B}'|$. On the other hand, the tree T' contains $\alpha+1$ vertices of H', that is, $|V(T'\cap H')| = \alpha+1$. Therefore, using Equations (2.1) and (2.3), we get that the total number of occurrences of the vertices in H' in some tree of $\mathcal{B}$ is

$$\begin{aligned}\sum_{v\in V(H')}|\{T\in\mathcal{B} : v\in T\}| &= \sum_{T\in\mathcal{B}'}|V(T)| + |V(T'\cap H')| = \sum_{T\in\mathcal{B}'}|E(T)| + |\mathcal{B}'| + \alpha + 1\\ &= \frac{nkC-1}{2} - \alpha + |\mathcal{B}'| + \alpha + 1 \geq \frac{nkC-1}{2} + \frac{nk-1}{2} + 1 + 1\\ &= nk\cdot\frac{C+1}{2} + 1 = n\cdot\left\lceil\frac{C+1}{C}\frac{\Delta}{2}\right\rceil + 1\ ,\end{aligned}$$

which implies that at least one vertex of H' appears in at least $\left\lceil\frac{C+1}{C}\frac{\Delta}{2}\right\rceil+1$ subgraphs, which is a contradiction to $\mathcal{B}$ being a C-edge-partition of G in which each vertex appears in at most $\left\lceil\frac{C+1}{C}\frac{\Delta}{2}\right\rceil$ subgraphs. The theorem follows. □

It turns out that Theorem 2.2 allows us to find the value of $M(3,\Delta)$ for any $\Delta\geq 3$ odd.

Corollary 2.3 (Case $C=3$) *For any $\Delta\geq 3$ odd, $M(3,\Delta) = \left\lceil\frac{2\Delta+1}{3}\right\rceil$.*

Proof: If $\Delta\equiv 1 \pmod 6$ or $\Delta\equiv 5 \pmod 6$, then by Corollary 2.1, $M(3,\Delta) = \left\lceil\frac{2\Delta}{3}\right\rceil = \left\lceil\frac{2\Delta+1}{3}\right\rceil$. Otherwise, if $\Delta\equiv 3 \pmod 6$, then by Theorem 2.2, $M(3,\Delta) = \left\lceil\frac{2\Delta}{3}\right\rceil + 1 = \left\lceil\frac{2\Delta+1}{3}\right\rceil$. □

2.4.3 Relation of $M(C,\Delta)$ with the linear C-arboricity

A result of Thomassen [190], which settled a conjecture of Bermond *et al.* [52], states that the edges of a cubic graph can be 2-colored such that each monochromatic component is a path of length at most 5. That is, in such a coloring (that can be seen as a partition into paths) each vertex appears in exactly 2 paths with at most 5 edges each. Therefore, combining this result with *(iii)* of Lemma 2.3 we deduce that $M(C,3) = 2$ for any $C\geq 5$.

Let us now discuss how these ideas can be extended to other values of C and Δ. A *linear C-forest* in a graph is a forest consisting of paths of length at most C. The *linear C-arboricity* of a graph G is the minimum number of linear C-forests required to partition $E(G)$, and is denoted by $la_C(G)$ [52]. Let $la_C(\Delta) = \max_{G\in\mathcal{G}_\Delta} la_C(G)$. Clearly $M(C,\Delta)\leq la_C(\Delta)$ for all C,Δ, since the paths in a *linear C-forest* are graphs with at most C edges. Therefore, the following upper bound given by Alon *et al.* [33] also applies to $M(C,\Delta)$.

Theorem 2.3 (Alon *et al.* [33]) *There is an absolute constant $\beta>0$ such that for $\sqrt{\Delta} > C\geq 2$,*

$$la_C(\Delta) \leq \frac{C+1}{C}\frac{\Delta}{2} + \beta\sqrt{C\Delta\log\Delta}. \tag{2.4}$$

It turns out that the first addend of the right-hand side of Equation (2.4) is equal to the lower bound of Proposition 2.1, so Theorem 2.3 provides an additive $O(\sqrt{C\Delta\log\Delta})$-approximation of $M(C,\Delta)$ for $\sqrt{\Delta} > C\geq 2$. Although we have improved this bound for $M(C,\Delta)$ in Sections 2.3 and 2.4.1, the relation between $M(C,\Delta)$ and $la_C(\Delta)$ is of theoretical interest by its own.

2.4.4 Case $\Delta = 3$, $C = 4$

As discussed in Section 2.4.3, $M(C,3) = 2$ for $C \geq 5$. On the other hand, Theorem 2.2 implies that $M(3,3) = 3$, so by *(ii)* and *(ii)* of Lemma 2.3 we have that $M(C,3) = 3$ for $C \leq 3$. Therefore, the interesting question is whether $M(4,3)$ equals 2 or 3. The remainder of this section is devoted to prove that $M(4,3) = 2$ (see Corollary 2.5). First we need a classical result concerning cubic graphs and an easy extension to cubic multigraphs.

Theorem 2.4 (Petersen [168]) *Any cubic bridgeless graph has a perfect matching.*

Corollary 2.4 *Any cubic bridgeless multigraph without self-loops has a perfect matching.*

Proof: Let G be a cubic multigraph without self-loops. We can assume that G has no triple edges, otherwise G has only 2 vertices and any of the 3 edges is a perfect matching. Consider the simple graph G' built from G as follows: for each digon $\{\{u,v\},\{u,v\}\}$, add 2 new vertices s_{uv} and t_{uv}, and replace the digon with the edges $\{u,s_{uv}\},\{u,t_{uv}\},\{v,s_{uv}\},\{v,t_{uv}\}$, and $\{s_{uv},t_{uv}\}$. By Theorem 2.4, G' has a perfect matching M'. We now construct a perfect matching M of G from M'. For each edge $e \in M'$ such that e was also an edge of G, put e in M'. For each digon $\{\{u,v\},\{u,v\}\}$ of G, if any of the pairs $\{\{u,s_{uv}\},\{v,t_{uv}\}\}$ or $\{\{u,t_{uv}\},\{v,s_{uv}\}\}$ is in M', put one of the copies of $\{u,v\}$ in M. Otherwise, $\{s_{uv},t_{uv}\}$ belongs to M' and we do nothing. It is easy to check that M is a perfect matching of G. □

We are ready to prove the main result of this section.

Theorem 2.5 *The edges of every almost 3-regular multigraph G without self-loops can be partitioned into a set $\mathcal{W} = \{W_1, W_2, \ldots, W_k\}$ of trails of length at most 4 such that each vertex appears as the midpoint of a trail.*

Proof: Suppose the theorem is false and let G be a counterexample with the minimum number of vertices. G is connected as otherwise, we can take the union of the partitions of its connected components, which exist by minimality of G.

Case 1: G contains a bridge $e = \{u,v\}$. Then $G - \{e\}$ has exactly two components: U containing u and V containing v. Without loss of generality, we may choose U to be the component with no degree 2 vertex in G and e is chosen so that U is maximal with this property. Thus this component U of $G - \{e\}$ is almost 3-regular (only u has degree 2). By minimality of G, U can be partitioned into a set $\mathcal{W}^u$ of trails as in the statement of the theorem.

If v has degree 2 in G then $V - \{v\}$ is almost 3-regular. By minimality of G, $V - \{v\}$ can be partitioned into a set $\mathcal{W}^v$ of trails as in the theorem. Now the only edges of G not in any trail in $\mathcal{W}^u \cup \mathcal{W}^v$ are those incident to v. Thus taking $\mathcal{W}^u \cup \mathcal{W}^v$ together with a trail consisting of the 2 edges incident to v (which has v as a midpoint) yields the required partition of the edges of G into trails. This contradicts the fact that G is a counterexample.

If v has degree 3 in G, let x, y be the neighbors of v in V (see Figure 2.2(a)). We can assume $x \neq y$ (i.e., $\{v, x\}$ and $\{v, y\}$ are not parallel edges) since otherwise, the third edge incident to $x = y$ is a cut edge whose choice (instead of e) would increase the size of U. Let H be the graph obtained from $V - \{v\}$ by adding an edge $f = \{x, y\}$ (see Figure 2.2(b)). By minimality of G, H can be partitioned into a set $\mathcal{W}^v$ of trails. We now attempt to transform $\mathcal{W}^u \cup \mathcal{W}^v$ into a partition of G into trails.

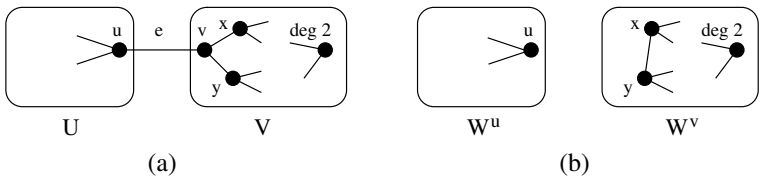

Figure 2.2: (a) A bridge $e = \{u, v\}$ in an almost 3-regular graph G with components U and V of $G - \{e\}$. (b) Graphs smaller than G from which we obtain a partition into trails W^u and W^v.

The edge f appears in some trail $\{W_1, \{x, y\}, W_2\}$ of $\mathcal{W}^v$, where W_1 is a (possibly empty) trail ending at x and W_2 is a (possibly empty) trail starting at y. At least one of the subtrails $\{W_1, \{x, y\}\}$ or $\{\{x, y\}, W_2\}$ has fewer than 3 edges. Without loss of generality, it is $\{W_1, \{x, y\}\}$. Replace this trail with $\{W_1, \{x, v\}, \{v, u\}\}$ which has length at most 4, and $\{\{v, y\}, W_2\}$ which has length less than or equal to $\{W_1, \{x, y\}, W_2\}$. Note that x and v are midpoints of the first trail and y is the midpoint of the second trail. Furthermore, any other vertex which was a midpoint in $\{W_1, \{x, y\}, W_2\}$ is still a midpoint (since W_1 and W_2 appear as subtrails).

Thus the union of $\mathcal{W}^u$ and $\mathcal{W}^v$ with the above replacement yields a partition of G into trails of length at most 4 with the desired property, which is a contradiction.

Case 2: G does not contains a bridge. If G is 3-regular, let $G' = G$. Otherwise, let G' be the graph obtained from G by replacing the vertex of degree 2 with an edge between its neighbors. Note that G' is 3-regular and contains no bridges. Therefore, by Corollary 2.4, G' contains a perfect matching $M \subseteq E(G')$.

Since G' is 3-regular, $G' - M$ is 2-regular. Thus, $G' - M$ is a union of disjoint cycles. We can orient the cycles of $G' - M$ so that each vertex v has exactly one edge e_v pointing towards v. For each edge $\{u, v\} \in M$, $W_{uv} = \{e_u, \{u, v\}, e_v\}$ is a trail of length 3 (see Figure 2.3). Note that $\mathcal{W} = \{W_{uv} \mid \{u, v\} \in M\}$ is a partition of the edges of G' into trails of length 3. Furthermore, every vertex u in the matching appears as the midpoint of the trail corresponding to the edge of the matching in which u appears. Since M is a perfect matching, every vertex appears as the midpoint of some trail in $\mathcal{W}$. Thus $G' \neq G$ as otherwise, we have constructed a partition as required by the theorem. So G has a vertex v of degree 2 which we replaced with an edge $e = \{x, y\}$ to obtain G'. Let $W = \{W_1, \{x, y\}, W_2\}$ be the trail in $\mathcal{W}$ containing e, and recall that W has length 3. Replacing W with $\{W_1, \{x, v\}, \{v, y\}, W_2\}$ in $\mathcal{W}$ yields a partition of $E(G)$ into trails of length at most 4, which is a contradiction. □

Note that the simple trees with some vertex of degree 3 and the digon with a pendant edge at each side are *not* allowed in the partition stated in Theorem 2.5, since these graphs cannot be thought of as trails. The following corollary settles the value of $M(4, 3)$.

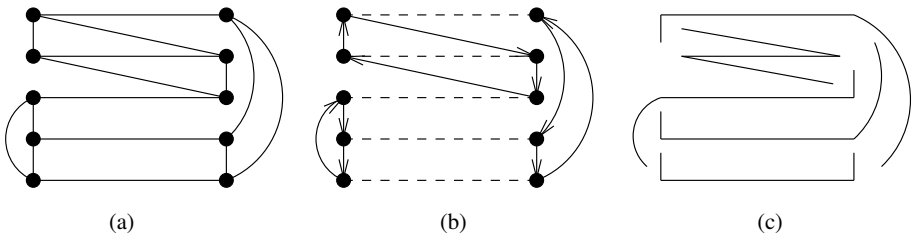

Figure 2.3: (a) A 3-regular graph G' with no bridges. (b) A matching M of G' (shown in dashed lines) and an orientation of the cycles of $G' - M$. (c) A partition of the edges of G' into trails of length 3 using M and the orientation of the cycle of $G' - M$ in (b).

Corollary 2.5 $M(4, 3) = 2$.

Proof: By Remark 2.2, we may restrict ourselves to 3-regular graphs. Thus, a 3-regular graph G is almost 3-regular and we may apply Theorem 2.5 to obtain a partition $\mathcal{W}$. Let $\mathcal{B} = \{E(W)\}_{W \in \mathcal{W}}$. Each vertex of G appears in at most two elements of $\mathcal{B}$, as G is 3-regular and each vertex appears as the midpoint of some trail in $\mathcal{W}$. □

2.4.5 Optimal construction for graphs with a perfect matching

In this section we focus on the case where the Δ-regular graphs are further restricted to contain a perfect matching. First observe that the proof of the general lower bound provided in Proposition 2.1 does not imply that the same lower bound carries over to Δ-regular graphs with a perfect matching. Indeed, the proof of Proposition 2.1 uses the existence of Δ-regular graphs with girth at least $C+1$, but those graphs may not necessarily contain a perfect matching. Fortunately, we can prove that the lower bound does not decrease when we assume that the graph contains a perfect matching.

Proposition 2.3 *Let $\Delta \geq 3$ be odd and let C be the class of Δ-regular graphs that contain a perfect matching. Then $M(C, \Delta, C) \geq \left\lceil \frac{C+1}{C} \frac{\Delta}{2} \right\rceil$ for all $C \geq 1$.*

Proof: We shall construct a Δ-regular graph G with a perfect matching and girth at least $C + 1$, and then the proof of Proposition 2.1 applied to G yields the desired bound. The details follow.

For any two positive integers Δ and C, Chandran provided in [67, Section 2.1] an explicit and simple construction of a graph H such that

- H has girth strictly greater than C;
- H contains a perfect matching (in fact, H is obtained from a perfect matching by adding the appropriate edges); and
- The degree of the vertices of H is either $\Delta - 2$, $\Delta - 1$, or Δ.

We will construct from H our Δ-regular graph G. Let $v_1, \ldots, v_{k_1}$ be the vertices of degree $\Delta-1$ in H, and let $v_{k_1+1}, \ldots, v_{k_1+k_2}$ be the vertices of degree $\Delta-2$ in H. Let F be a (k_1+2k_2)-regular graph with girth at least $C+1$ (which exists by the result of Erdös and Sachs [111]), and let $f = |V(F)|$. Let the vertices of F be $u^1, \ldots, u^f$. To construct G, first make f copies of H, and let v_i^j be the copy of vertex v_i in the j-th copy of H, for $i = 1, \ldots, k_1 + k_2$ and $j = 1, \ldots, f$. Intuitively, each copy of H corresponds to a vertex of F. We now add $|E(F)|$ edges among the f copies of H as follows. We assign labels from 1 to $k_1 + 2k_2$ to the vertices of H with degree less than Δ in the following way: for $i = 1, \ldots, k_1$, vertex v_i gets label i, and for $i = k_1 + 1, \ldots, k_1 + k_2$, vertex v_i gets labels i and $k_2 + i$. For each vertex of F, we label arbitrarily the edges incident to it with distinct integers from 1 to $k_1 + 2k_2$ (recall that F is $(k_1 + 2k_2)$-regular). This way, each edge of F gets two labels, one from each end-vertex. Then, for each edge $\{u^{j_1}, u^{j_2}\} \in E(F)$ with labels (ℓ_1, ℓ_2), we add an edge between the vertices labeled ℓ_1 and ℓ_2 in the j_1-th and j_2-th copies of H, respectively.

This completes the construction of G. Note that the copies of the vertices that had degree Δ in H have also degree Δ in G. Since one (resp. two) edges have been added to each vertex of degree $\Delta - 1$ (resp. $\Delta - 2$), it is clear that G is Δ-regular. Since each copy of H had a perfect matching and no edge of any copy of H has been removed, G has also a perfect matching. Finally, the girth of G is at least $C + 1$. Indeed, the girth of each copy of H is at least $C + 1$ by [67]. Therefore, each cycle c of length at most C in G should visit strictly more than one copy of H. By the construction of G, such a cycle c in G would induce a cycle of length at most C in F among the vertices corresponding to the copies of H visited by c. But this is impossible as the girth of F is at least $C + 1$. □

We are now ready to provide an optimal construction for all $\Delta \geq 3$ odd and $C \geq 1$ when the request graph is restricted to have a perfect matching.

Proposition 2.4 *Let $\Delta \geq 3$ be odd and let $\mathcal{C}$ be the class of Δ-regular graphs that contain a perfect matching. Then $M(C, \Delta, \mathcal{C}) = \left\lceil \frac{C+1}{C} \frac{\Delta}{2} \right\rceil$ for all $C \geq 1$.*

Proof: The lower bound follows from Proposition 2.3. To prove the upper bound, let G be a Δ-regular with a perfect matching M. Then $G - M$ is $(\Delta - 1)$-regular, with $\Delta - 1$ even. We orient the edges of $G - M$ in an Eulerian tour, and assign to each vertex $v \in V(G)$ its $\frac{\Delta-1}{2}$ out-edges E_v^+. We distinguish three cases.

(1) $\Delta < C$**.** For each edge $\{u, v\} \in M$, build a tree with Δ edges consisting of $\{u, v\}$, $\frac{\Delta-1}{2}$ edges from E_u^+, and $\frac{\Delta-1}{2}$ edges from E_v^+. The number of occurrences of each vertex is $1 + \Delta - \frac{\Delta+1}{2} = \frac{\Delta+1}{2}$. The lower bound equals $\left\lceil \frac{C+1}{C} \frac{\Delta}{2} \right\rceil = \frac{\Delta-1}{2} + \left\lceil \frac{1}{2} + \frac{\Delta}{2C} \right\rceil$, which equals $\frac{\Delta+1}{2}$ as $\Delta < C$.

(2) $\Delta \geq C$ and $C \geq 3$ is odd (the case $C = 1$ is trivial by Lemma 2.3). For each edge $\{u, v\} \in M$, build a tree with C edges consisting of $\{u, v\}$, $\frac{C-1}{2}$ edges from E_u^+, and $\frac{C-1}{2}$ edges from E_v^+. Partition the remaining $\frac{\Delta-1}{2} - \frac{C-1}{2} = \frac{\Delta-C}{2}$ edges assigned to each vertex into $\left\lceil \frac{\Delta-C}{2C} \right\rceil$ stars with at most C edges. The number of occurrences of each vertex is

$$1 + \left\lceil \frac{\Delta - C}{2C} \right\rceil + \Delta - \frac{\Delta + 1}{2} = \left\lceil \frac{C+1}{C} \frac{\Delta}{2} \right\rceil.$$

(3) $\Delta \geq C$ and $C \geq 4$ is even (the case $C = 2$ is solved by Corollary 2.2). Build a tree with $C - 1$ edges consisting of $\{u, v\}$, $\frac{C-2}{2}$ edges from E_u^+, and $\frac{C-2}{2}$ edges from E_v^+. Partition the remaining $\frac{\Delta-1}{2} - \frac{C-2}{2} = \frac{\Delta-C+1}{2}$ edges assigned to each vertex into stars with at most C edges. The number of occurrences of each vertex is

$$1 + \left\lceil \frac{\Delta - C + 1}{2C} \right\rceil + \frac{\Delta - 1}{2} = \left\lceil \frac{\Delta(C+1) + 1}{2C} \right\rceil = \left\lceil \frac{C+1}{C} \frac{\Delta}{2} \right\rceil,$$

where the last equality holds because both Δ and $(C + 1)$ are odd. □

2.4.6 Towards a proof for the remaining cases

In this section, we describe an attempt to prove that the lower bound $\left\lceil \frac{C+1}{C} \frac{\Delta}{2} \right\rceil$ of Proposition 2.1 is attained in the remaining cases where $\Delta \geq 5$ is odd. We attempt to resolve the remaining cases by using induction and Tutte's matching theorem. We may assume that Δ is odd by Theorem 2.1.

The idea is to use the construction from Proposition 2.4 of Section 2.4.5, which solves the case where the graph contains a perfect matching, as a base case for a proof by induction. Then, if the graph does not contain a perfect matching, an easy consequence of Tutte's matching theorem shows that it contains an edge-cut of size at most $\Delta - 1$. We would then like to recurse on each side of the cut as we did in the proof of Theorem 2.5 and combine the edge-partitions of each side into a partition of the whole graph. However, as opposed to Theorem 2.5, it is more difficult to deal with the edges across the cut in this case.

We may orient each edge $e = \{u, v\}$ across the cut from u to v and let the side containing u decide which partition will contain e. To guarantee that v is not incident to too many subgraphs at the end, we can simply force v to be incident to one fewer subgraph in the edge-partition of the side containing v.

We note that, since the cuts have size less than Δ, it is possible to recursively orient the edges of cuts so that no side has more than $\Delta - 1$ edges pointing towards it (including edges from previous steps of the recursion).

However, it seems difficult to control the distribution of the edges pointing towards a side. If, for example, a single vertex v had $\Delta - 1$ edges pointing towards it, then it is clearly impossible to obtain the desired edge-partition, as v would need to be in a negative number of parts. On the other hand, if it were possible to control the distribution of the edges pointing towards a side, the following strengthening of Proposition 2.4 would be sufficient to prove the base case of the induction.

Definition 2.2 *G is* near-Δ-regular *if the vertices of G have degrees between $\frac{\Delta}{2}$ and Δ and $|E(G)| \geq \frac{\Delta}{2}(|V(G)| - 1) - 1$ (i.e., the total degree is off by at most $\Delta - 1$).*

Lemma 2.4 *Let $LB(C, \Delta) = \left\lceil \frac{C+1}{C} \frac{\Delta}{2} \right\rceil$, the lower bound of Proposition 2.1. Let C, Δ be positive integers with Δ odd and $(\Delta - 1)/2$ not a multiple of C. Let G be a near-Δ-regular graph with girth at least 5 and a perfect matching. Then G has an edge-partition where each vertex v is incident to at most $LB(C, \Delta) - (\Delta - \mathbf{deg}(v))$ subgraphs of the partition.*

We note that it may be possible to first recursively find all the cuts and then orient the edges so that no vertex has more than $(\Delta - 1)/2$ edges pointing towards it. We also note that the above lemma is not in its strongest form (e.g., we could have the total degree differ by more the one stated in the lemma) but we clearly cannot relax the condition that every vertex v has degree greater than $\Delta/2$ (otherwise, v is contained in too many subgraphs even if we use stars of size C centered at v). We now prove Lemma 2.4.

Proof: Let M be a perfect matching in G. Since at most Δ vertices of G have degree not equal to Δ, at most Δ edges in M connect an odd degree vertex to an even degree vertex. Let G' be the graph obtained from G by removing edges of M matching odd degree vertices of G and adding (at most $\Delta/2$) edges to pair up the remaining odd degree vertices of G. Thus, G' is an even graph and we may obtain an Eulerian orientation O' of G'.

O' induces an orientation of some of the edges of G. We orient the remaining edges of G "both ways" and count half towards the in-degree and half towards the out-degree of the vertex. Let S be the set of vertices of G with degree less than Δ. We reverse some of the arcs of O so that all vertices in S have out-degree at least $\Delta/2$. This can be done greedily since G has no C_4 and we are only off by $\Delta - 1$ from the total degree. We call this new orientation O.

Let S' be the set of vertices with out-degree less than $\Delta/2$ in O. Note that the vertices of S' have degree Δ and out-degree at least $(\Delta - 3)/2$. Let $N^-(v)$ denote the set of vertices with an arc to v in O. Note that for two distinct vertices u and v in G, their neighborhood intersects in at most one vertex (since G has girth at least 5). Therefore $N^-(u) \cap N^-(v)$ also contains at most one vertex.

Therefore, we may find a subgraph H of the graph induced by the edges $N^-(v)$ to v for all v in S' with the following properties:

- Each vertex in $N^-(S')$ has degree at most 1.
- Each vertex in $s \in S'$ has degree at least $\delta^-(s) - (\Delta - 1)/2 \geq \delta^-(s) - (2C - 1)$, where $\delta^-(s)$ is the in-degree of s.

We say that a star or double-star is *full* if it contains exactly C edges.

Now, we can find a set $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3$ of edge disjoint subgraphs of G such that

- $\mathcal{S}_1$ is a set of full double stars centered at the endpoints of unoriented edges in O,
- $\mathcal{S}_2$ is a set of full stars,
- $\mathcal{S}_3 = \{S_v\}_{v \in V(G)}$, where S_v is a star centered at v of size at most $C - 1$, and
- only stars in $\mathcal{S}_3$ contain edges of H.

These edge disjoint subgraphs can be found greedily by first finding $\mathcal{S}_1$ and then partitioning the out-edges of each vertex into sets of size C and a remainder set of edges of size $\leq C$.

Now, for each $s \in S'$, remove all but two stars centered at s in $\mathcal{S}_2$ and remove one star centered at s in $\mathcal{S}_3$. Let R be the set of edges removed in this way. For each edge

$e = \{u, v\} \in E(H)$, add an out-edge of v in R if there is any left (and remove this edge from R) to the star containing e. By the properties of H, no edges of R are left in the end.

We claim that this new set of subgraphs form a C-edge-partition where each vertex v is incident to at most $LB(C, \Delta) - (\Delta - \mathbf{deg}(v))$ partitions.

Indeed, the elements of $\mathcal{S}'$ are edge disjoint and have size at most C (since every star in $\mathcal{S}_3$ has size at most $C - 1$). The vertices $v \in V - S''$ are incident to

$$\begin{aligned}
\frac{\Delta + 1}{2C} + \mathbf{deg}(v) - \frac{\Delta + 1}{2} &= \frac{\Delta + 1}{2C} + \frac{\Delta - 1}{2} - (\Delta - \mathbf{deg}(v)) \\
&= \frac{\Delta(C + 1) + 1 - C}{2C} - (\Delta - \mathbf{deg}(v)) \\
&\leq LB(C, \Delta) - (\Delta - \mathbf{deg}(v))
\end{aligned}$$

subgraphs if v is not incident to an unoriented edge, and

$$\begin{aligned}
1 + \left\lceil \frac{\Delta - C}{2C} \right\rceil + \mathbf{deg}(v) - \frac{\Delta + 1}{2} &= 1 + \left\lceil \frac{\Delta - C}{2C} \right\rceil + \frac{\Delta - 1}{2} - (\Delta - \mathbf{deg}(v)) \\
&= \left\lceil \frac{\Delta + C + C(\Delta - 1)}{2C} \right\rceil - (\Delta - \mathbf{deg}(v)) \\
&= \left\lceil \frac{(C + 1)\Delta}{2C} \right\rceil - (\Delta - \mathbf{deg}(v)) \\
&= LB(C, \Delta) - (\Delta - \mathbf{deg}(v))
\end{aligned}$$

subgraphs if v is incident to an unoriented edge. This satisfies the conditions in the lemma.

Recall that vertices in S' have out-degree at least $(\Delta - 3)/(2C)$. If $v \in S'$ and v is incident to an unoriented edge, v appears in

$$\begin{aligned}
\frac{\Delta - 3 - (2C - 2)}{2C} + \mathbf{deg}(v) - \frac{\Delta - 3}{2} &= \frac{\Delta - 1 - 2C}{2C} + \mathbf{deg}(v) - \frac{\Delta + 1}{2} + \frac{4}{2} \\
&= \frac{\Delta - 1}{2C} - 2 + \mathbf{deg}(v) - \frac{\Delta + 1}{2} + 2 \\
&\leq \frac{\Delta + 1}{2C} + \mathbf{deg}(v) - \frac{\Delta + 1}{2} \\
&\leq LB(C, \Delta) - (\Delta - \mathbf{deg}(v))
\end{aligned}$$

subgraphs. If $v \in S'$ and and v is not incident to an unoriented edge, v appears in

$$\begin{aligned}
1 + \left\lceil \frac{\Delta - 3 - C - (2C - 2)}{2C} \right\rceil + \mathbf{deg}(v) - \frac{\Delta - 3}{2} &= 1 + \left\lceil \frac{\Delta - 1 - C}{2C} \right\rceil - 2 + \mathbf{deg}(v) - \frac{\Delta + 1}{2} + 2 \\
&\leq 1 + \left\lceil \frac{\Delta - C}{2C} \right\rceil + \mathbf{deg}(v) - \frac{\Delta + 1}{2} \\
&= LB(C, \Delta) - (\Delta - \mathbf{deg}(v))
\end{aligned}$$

subgraphs. Again, the conditions in the lemma are satisfied as required.

□

2.5 Conclusions

In this chapter we introduced the traffic grooming problem in unidirectional WDM rings when the request graph belongs to the class of graphs with maximum degree Δ. Such a model allows the network to support dynamic traffic without reconfiguring the electronic equipment. We showed that this problem is essentially equivalent to finding the least integer $M(C, \Delta)$ such that the edges of any graph with maximum degree at most Δ can be partitioned into subgraphs with at most C edges and each vertex appears in at most $M(C, \Delta)$ subgraphs. We established the value of $M(C, \Delta)$ for many cases, leaving open only the case where $\Delta \geq 5$ is odd, $\Delta \pmod{2C}$ is between 3 and $C - 1$, $C \geq 4$, and the graph does not contain a perfect matching. Table 2.1 summarizes what is known about $M(C, \Delta)$, including the case where the graph has a perfect matching. For the remaining cases, we hope to either extend the counterexample given in Section 2.4.2 or to complete the partial proof given in Section 2.4.6, which can be seen as a strengthening of Proposition 2.4.

Considering bounded-degree request graphs is natural from a networking perspective. It would be also interesting to consider as input other families of request graphs that make sense from a telecommunications point of view, like circulant graphs or graphs of bounded diameter.

$C \vert \Delta$	1	2	3	4	5	6	7	8	9	...	Δ even	Δ odd
1	1	2	3	4	5	6	7	8	9	...	Δ	Δ
2	1	2	3	3	4	5	6	6	7	...	$\lceil \frac{3\Delta}{4} \rceil$	$\lceil \frac{3\Delta}{4} \rceil$
3	1	2	3 (2)	3	4	5 (4)	5	6	7 (6)	...	$\lceil \frac{2\Delta}{3} \rceil$	$\lceil \frac{2\Delta+1}{3} \rceil$ ($\lceil \frac{2\Delta}{3} \rceil$)
4	1	2	2	3	4	4	5	5	6	...	$\lceil \frac{5\Delta}{8} \rceil$	$\geq \lceil \mathbf{\frac{5\Delta}{8}} \rceil$ (=)
5	1	2	2	3	4 (3)	4	5	5	6	...	$\lceil \frac{3\Delta}{5} \rceil$	$\geq \lceil \mathbf{\frac{3\Delta}{5}} \rceil$ (=)
6	1	2	2	3	$\geq$ **3** (=)	4	5	5	6	...	$\lceil \frac{7\Delta}{12} \rceil$	$\geq \lceil \mathbf{\frac{7\Delta}{12}} \rceil$ (=)
7	1	2	2	3	$\geq$ **3** (=)	4	5 (4)	5	6	...	$\lceil \frac{4\Delta}{7} \rceil$	$\geq \lceil \mathbf{\frac{4\Delta}{7}} \rceil$ (=)
8	1	2	2	3	$\geq$ **3** (=)	4	$\geq$ **4** (=)	5	6	...	$\lceil \frac{9\Delta}{16} \rceil$	$\geq \lceil \mathbf{\frac{9\Delta}{16}} \rceil$ (=)
9	1	2	2	3	$\geq$ **3** (=)	4	$\geq$ **4** (=)	5	6 (5)	...	$\lceil \frac{5\Delta}{9} \rceil$	$\geq \lceil \mathbf{\frac{5\Delta}{9}} \rceil$ (=)
...	...	...	...	...	...	...	...	...	...	...	...	...
C	1	2	2	3	$\geq$ **3** (=)	4	$\geq$ **4** (=)	5	$\geq$ **5** (=)	...	$\lceil \frac{C+1}{C} \frac{\Delta}{2} \rceil$	$\geq \lceil \mathbf{\frac{C+1}{C}} \frac{\Delta}{2} \rceil$ (=)

Table 2.1: Known values of $M(C, \Delta)$. The **bold** cases remain open. The cases in brackets only hold if the graph has a perfect matching. The symbol "(=)" means that the corresponding lower bound is attained.

Chapter 3

Bidirectional WDM Rings

In this chapter we study the minimization of ADMs (Add-Drop Multiplexers) in optical WDM bidirectional rings considering symmetric shortest path routing and all-to-all unitary requests. We precisely formulate the problem in terms of graph decompositions, and state a general lower bound for all the values of the grooming factor C and n, the size of the ring. We first study exhaustively the cases $C = 1$, $C = 2$, and $C = 3$, providing improved lower bounds, optimal constructions for several infinite families, as well as asymptotically optimal constructions and approximations. We then study the case $C > 3$, focusing specifically on the case $C = k(k+1)/2$ for some $k \geq 1$. We give optimal decompositions for several congruence classes of n, using the existence of a certain combinatorial design. We conclude with a comparison of the switching cost in unidirectional and bidirectional WDM rings.

Keywords: traffic grooming, SONET ADM, optical WDM network, graph decomposition, combinatorial designs.

3.1 Introduction

Whereas traffic grooming in unidirectional rings has been widely studied in the literature (see page 30), the bidirectional ring has not been studied from a graph partitioning approach. Filling this gap in the literature is the main objective of this chapter.

Namely, we focus on bidirectional rings with symmetric shortest path routing, and on the all-to-all case. We begin by formally stating the problem in terms of graph partitioning in Section 3.1.1. In Section 3.2 we provide lower bounds that improve those existing in the literature. The remainder of the chapter is devoted to find families of solutions for certain values of C and n. Namely, in Section 3.3 we show that the case $C = 1$ is relatively easy to solve. In Section 3.4 we study the case $C = 2$, improving the general lower bound and providing a $\frac{34}{33}$-approximation for all values of n. In Section 3.5 we tackle the case $C = 3$, improving the general lower bound and giving families of optimal and asymptotically

optimal solutions for all values of n. In Section 3.6 we study the case $C > 3$, by either improving the general lower bound or providing families of optimal solutions when C is of the form $1+2+\ldots+k$. In Section 3.7 we compare the lower bounds for the switching cost in unidirectional and bidirectional rings. We conclude the chapter in Section 3.8. We first state precisely the problem we study.

3.1.1 Statement of the problem

Load constraint. In a graph-theoretical approach, we are given an optical network represented by a directed graph G on n vertices (in many cases a symmetric one) – called the *physical graph* –, for example a unidirectional ring $\vec{C}_n$ or a bidirectional symmetric ring C_n^*. We are given also a traffic (or instance) matrix, that is a family of connection requests represented by an arc-weighted multidigraph I – called the *logical* or *request graph* – where the number of arcs from i to j corresponds to the number of requests from i to j, and the weight of each arc corresponds to the amount of bandwidth used by each request. Here we suppose that there is exactly one request from i to j (all-to-all case) and that each request uses the same bandwidth. In that case $I = K_n^*$. We also suppose that the bandwidth used by any request is a fraction $1/C$ of the available bandwidth of a wavelength. Said otherwise, each wavelength λ can carry on a given arc at most C requests. This positive integer C is called the *grooming factor*. For a wavelength λ, we denote by B_λ the set of requests carried by λ. Satisfying a request r from i to j consists in finding a dipath $P(r)$ in G and assigning it a wavelength λ. Note that a wavelength λ is directed either clockwise or counterclockwise, so all the dipaths associated to requests in a same B_λ are directed in the same way.

For a subgraph B_λ of requests of I, we define the *load* of an arc e of G, $L(B_\lambda, e)$, as the number of requests which are routed through e, that is

$$L(B_\lambda, e) := |\{P(r); r \in E(B_\lambda); e \in P(r)\}|.$$

Note that if B_λ is associated to a clockwise (resp. counterclockwise) wavelength λ, only the clockwise (resp. counterclockwise) arcs of the ring are loaded by B_λ. The constraint given by the grooming factor C means that for each subgraph B_λ and each arc e, $L(B_\lambda, e)$ is at most C. In this chapter we focus on the bidirectional ring topology with all-to-all unitary requests. Therefore, our problem consists in finding a partition of K_n^* into subdigraphs B_λ satisfying the load constraint for C_n^* and such that the total number of vertices is minimized. We have two choices for routing a request (i, j): either clockwise or counterclockwise. Although there is no physical constraint imposing it, it is common for the operator to consider symmetric routings. That is, if the request (i, j) is routed clockwise, then the request (j, i) is routed counterclockwise. Furthermore it is also common for the sake of simplicity to use shortest path routing. Therefore we will restrict ourselves to symmetric shortest path routings. Let us see how the restrictions on the routing affect the solutions.

Constraints on the routing. In a ring C_n^* with an odd number of vertices, shortest path routing implies symmetric routing. But in a ring with an even number of vertices

this is not necessarily the case, as a request of the form $(i, i + \frac{n}{2})$ can be routed via a shortest path in both directions. Consider for example $n = 4$ and $C = 2$. If we do not impose symmetric routing, we can have a solution consisting of the two subdigraphs B_{λ_1} with the requests $(0,1)$, $(1,2)$, $(2,3)$, $(3,0)$, $(0,2)$, and $(2,0)$ routed clockwise, and B_{λ_2} with the requests $(1,0)$, $(0,3)$, $(3,2)$, $(2,1)$, $(1,3)$, and $(3,1)$ routed counterclockwise. Altogether we use 8 ADMs. Suppose now that we further impose symmetric routing, and assume without loss of generality that the requests $(0,2)$ and $(1,3)$ are routed clockwise. The best we can do for a B_λ with 4 vertices is to put 5 requests if λ is clockwise, namely $(0,1)$, $(1,2)$, $(2,3)$, $(3,0)$, and at most one of $(0,2)$ and $(1,3)$. The other request out of $(0,2)$ and $(1,3)$ will need 2 ADMs, so we use a total of 12 ADMs. If we do not use any B_λ with 4 vertices, note that a subdigraph with 3 (resp. 2) vertices contains at most 3 requests (resp. 1 request). Therefore to route all the requests we need at least 12 ADMs.

Imposing shortest path routing might increase the number of ADMs of an optimal solution. Consider for example $n = 3$ and $C = 3$. With shortest path routing, we need two subdigraphs B_{λ_1} with the requests $(0,1)$, $(1,2)$, $(2,0)$ and B_{λ_2} with the requests $(1,0)$, $(2,1)$, $(0,2)$, for a total of 6 ADMs (each arc of C_3^* is loaded once). Without the constraint of shortest path routing, we can do it with 3 ADMs, namely with all the requests routed clockwise. In that case, the requests $(1,0)$, $(2,1)$, and $(0,2)$ are routed via dipaths of length 2 (for instance, the request $(1,0)$ used the arcs $(1,2)$ and $(2,0)$). In that case the load of the arcs (in the clockwise direction) is 3.

We cannot always use shortest path routing and have a minimum load. Indeed, consider the case $C = 1$ and a set of 3 requests (i,j), (j,k), and (k,i) forming a triangle. The subdigraph formed by the 3 requests routed in the same direction has load 1, but there is not reason that the associated routes are shortest paths. For example, let $n = 5$ and $(0,1)$, $(1,2)$, $(2,0)$ be the three mentioned requests, which we assume to be routed clockwise. If we want a valid solution, then the request $(2,0)$ is routed via the path $[2,3,4,0]$ of length 3 (and not 2). If we want to use shortest paths, then these three requests induce load 2, hence they cannot fit together in the same wavelength. Summarizing, in this example either we use shortest paths and the load is 2 or we get a solution with load one but not using shortest paths.

Symmetric shortest path routing. However, in the sequel of the chapter we will only consider symmetric shortest path routing. Besides being a common scenario in telecommunication networks, this assumption also simplifies the problem, as we can split it into two separate problems, half of the requests being routed clockwise and half counterclockwise. Each of these two subproblems can be viewed as a grooming problem where $G = \vec{C}_n$ (the unidirectional cycle) and $I = T_n$, where T_n is a tournament on n vertices, that is, a complete oriented graph (for each pair of vertices $\{i,j\}$ there is exactly one of the arcs (i,j) or (j,i)).

As we consider shortest path routing, for n odd T_n is unique. But for n even we have two possibilities for the pairs of the form $\{i, i + \frac{n}{2}\}$: either the arc $(i, i + \frac{n}{2})$ or $(i + \frac{n}{2}, i)$. So the choice of these arcs has to be made. We are ready to state precisely our problem.

Traffic Grooming in Bidirectional Rings
Input: A unidirectional cycle $\vec{C}_n$ with vertices $0, \ldots, n-1$, a grooming factor C and a digraph of requests consisting of the tournament T_n with arcs $(i, i+1)$ for $0 \le i \le n-1$ and $1 \le q \le \frac{n-1}{2}$, plus if n is even $\frac{n}{2}$ arcs of the form $(i, i+\frac{n}{2})$, where we cannot have both $(i, i+\frac{n}{2})$ and $(i+\frac{n}{2}, i)$ (or said otherwise, for n even we have one of the two arcs $(i, i+\frac{n}{2})$ or $(i+\frac{n}{2}, i)$ for $0 \le i \le \frac{n}{2}-1$).
Output: A partition of T_n into digraphs B_λ, $1 \le \lambda \le \Lambda$, such that for each arc $e \in E(\vec{C}_n)$, $L(B_\lambda, e) \le C$.
Objective: Minimize $\sum_{\lambda=1}^{\Lambda} |V(B_\lambda)|$. The minimum will be denoted $A(C, n)$.

Remark 3.1 *Solutions to the original problem can be found by solving the above problem and using the solution for the counterclockwise requests by reversing the orientation of the arcs of $\vec{C}_n$ and T_n. Therefore, the total number of ADMs for the original problem – under the constraints of symmetric shortest path routing – is $2A(C, n)$.*

Let us see an example for $n = 5$ and $C = 1$. Then the following three subdigraphs form a solution with 10 ADMs: one with arcs $(0,1),(1,3),(3,0)$, another with arcs $(1,2),(2,4),(4,1)$, and another with arcs $(0,2),(2,3),(3,4),(4,0)$. A solution for the bidirectional ring C_5^* and $I = K_n^*$ uses 20 ADMs.

Let now $n = 5$ and $C = 2$. We can use the preceding solution or another one with also 10 ADMs with only two $\vec{C}_5$'s with arcs $(0,2),(1,2),(2,3),(3,4),(4,5)$ and $(0,2),(2,4),(4,1),(1,3),\ (3,0)$, the second one inducing load 2. But we can do better, with only 8 ADMs, with one subdigraph with arcs $(1,3),(3,4),(4,1)$, and another one with arcs $(0,1),(1,2),(0,2),(2,3),\ (2,4),(3,0),(4,0)$. This latter partition is optimal.

To tackle our problem we will use tools from design theory, similar to those used for the unidirectional ring and $I = K_n$ [47, 48]. In particular, it is helpful to use for a given C digraphs having a maximum ratio number of arcs over number of vertices (see Section 3.2.2).

Admissible digraphs. Let $B_\lambda = (V_\lambda, E_\lambda)$ be a digraph with $V_\lambda = \{a_0, \ldots, a_{p-1}\}$ involved in a partition of the tournament T_n. Note that the edges of B_λ belong to T_n, so $(a_i, a_j) \in E_\lambda$ if and only $d_{\vec{C}_n}(a_i, a_j) \le \frac{n}{2}$, where $d_{\vec{C}_n}(a_i, a_j)$ is the distance between a_i and a_j in $\vec{C}_n$.

A digraph B_λ is said to be *admissible* if it satisfies the load constraint, that is, $L(B_\lambda, e) \le C$ for each arc $e \in E(\vec{C}_n)$. A partition of T_n into admissible subdigraphs is called *valid*. As the paths associated to an arc of B_λ form a dipath (an interval) in $\vec{C}_n$, the load is exactly the same as if we consider B_λ embedded in a cycle $\vec{C}_p$ with vertex set $0, 1, \ldots, p-1$. More precisely, we associate to B_λ the digraph B_λ^p with vertices $0, 1, \ldots, p-1$ and with $(i, j) \in E(B_\lambda^p)$ if and only if $(a_i, a_j) \in E(B_\lambda)$. Hence, to compute the load we will consider digraphs with p vertices and their load in the associated $\vec{C}_p$. Note that it can happen that $d_{\vec{C}_n}(a_i, a_j) \le \frac{n}{2}$ but $d_{\vec{C}_p}(i, j) > \frac{p}{2}$, and viceversa.

Figure 3.1(a) illustrates a digraph B_λ that is admissible for $n = 8$ and $C = 2$, as it induces load 2 in $\vec{C}_8$. Its associated digraph B_λ^4 is shown in Figure 3.1(b). Figure 3.1(c) shows a digraph B'_λ which has also B_λ as associated digraph, but it is not admissible as (a_3, a_0) is not an arc of T_8.

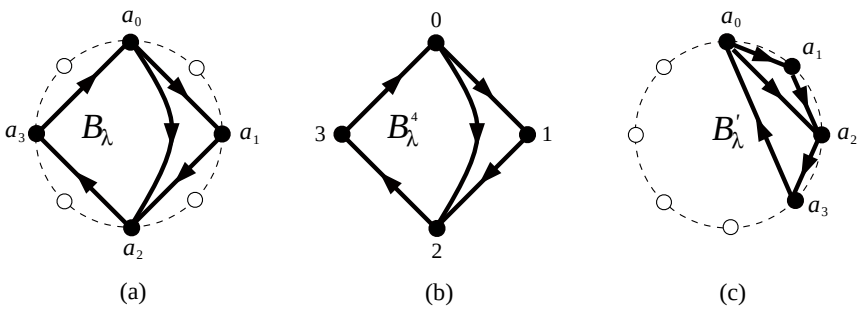

Figure 3.1: (a) Digraph B_λ admissible for $n = 8$ and $C = 2$; (b) Its associated digraph B^4_λ; (c) Non-admissible digraph B'_λ that has also B^4_λ as associated digraph.

Figure 3.2(a) shows and admissible digraph for $n = 7$ and $C = 2$. Its associated digraph B^5_λ, which is depicted in Figure 3.2(b), induces load 2 but the arc $(1, 4)$ is not routed via a shortest path (although the arc (a_1, a_4) was in B_λ).

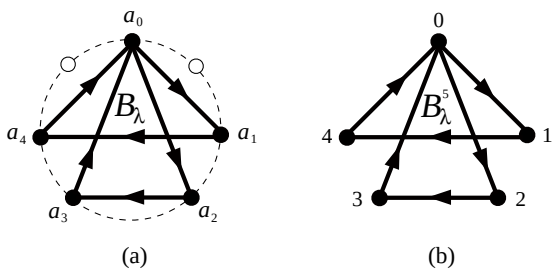

Figure 3.2: (a) Digraph B_λ admissible for $n = 7$ and $C = 2$; (b) Its associated digraph B^5_λ.

In what follows we will compute the load in the associated digraph, but we will have to be careful that the arcs of B_λ are those of T_n, as pointed out by the above examples.

3.2 Lower Bounds

In this section we state general lower bounds on the number of ADMs used by any solution.

3.2.1 Equations of the problem

Given a valid solution of the problem, let a_p denote the number of subgraphs of the partition with exactly p nodes, let A denote the total number of ADMs, let Λ denote the number of subgraphs of the partition, and let E_λ be the set of arcs of B_λ. Recall that here

$I = T_n$, which has $\frac{n(n-1)}{2}$ arcs. The following equalities hold:

$$A = \sum_{p=2}^{n} p a_p \tag{3.1}$$

$$\sum_{p=2}^{n} a_p = \Lambda \tag{3.2}$$

$$\sum_{\lambda=1}^{\Lambda} |E_\lambda| = \frac{n(n-1)}{2} \tag{3.3}$$

Proposition 3.1 *For $I = T_n$,*

$$\Lambda \geq \left\lceil \frac{n^2 + \alpha}{8C} \right\rceil, \text{ where } \alpha = \begin{cases} -1, & \text{if } n \text{ is odd} \\ 4, & \text{if } n \equiv 2 \pmod 4 \\ 8, & \text{if } n \equiv 0 \pmod 4 \end{cases}$$

Proof: The set of arcs of T_n of the form $(i, i+q)$, $0 \leq q < \frac{n}{2}$, load each arc of the ring exactly q times. So if n is odd the load of any arc of the ring is $1 + 2 + \ldots + \frac{n-1}{2} = \frac{n^2-1}{8}$.

If n is even the load due to these arcs is $1 + 2 + \ldots + \frac{n-2}{2} = \frac{n^2-2n}{8}$. We have to add the load due to arcs of T_n of the form $\left(i, i + \frac{n}{2}\right)$. As there are $\frac{n}{2}$ such arcs, the total load is $\frac{n^2}{4}$ and so one arc of the ring has load at least $\frac{n}{4}$.

If $n \equiv 2 \pmod 4$ that gives a load at least $\left\lceil \frac{n}{4} \right\rceil = \frac{n+2}{4}$, so one arc has load at least $\frac{n^2-2n}{8} + \frac{n+2}{4} = \frac{n^2+4}{8}$.

If $n \equiv 0 \pmod 4$ the maximum load due to the arcs $\left(i, i + \frac{n}{2}\right)$ is at least $\frac{n}{4}$, but in this case we can give a better bound. Indeed, suppose w.l.o.g. that we have the arc $\left(0, \frac{n}{2}\right)$, and let j be the number of arcs starting in the interval $[1, \frac{n}{2} - 1]$ of the form $\left(i, i + \frac{n}{2}\right)$ with $0 < i < \frac{n}{2}$. The load of the arc $\left(\frac{n}{2} - 1, \frac{n}{2}\right)$ of the ring is then $j + 1$. As there are $\frac{n}{2} - 1 - j$ arcs ending in the interval $[1, \frac{n}{2} - 1]$, the load of the arc $(0, 1)$ is $1 + \frac{n}{2} - 1 - j$. Therefore the sum of the loads of the arcs $(0, 1)$ and $\left(\frac{n}{2} - 1, \frac{n}{2}\right)$ is $\frac{n}{2} + 1$, and so one of these 2 arcs has load $\left\lceil \frac{n}{4} + \frac{1}{2} \right\rceil = \frac{n}{4} + 1$. The total load of this arc is $\frac{n^2-2n}{8} + \frac{n}{4} + 1 = \frac{n^2+8}{8}$.

As each subgraph can load one arc at most C times, we obtain the lemma. □

3.2.2 The parameter $\gamma(C, p)$

To obtain accurate lower bounds we need to bound the value of $|E_\lambda|$ for a digraph with $|V_\lambda| = p$ vertices, satisfying the load constraint (admissible digraph). As we discussed in the preceding section, we need only to consider the associated digraph embedded in $\vec{C}_p$. To this end, we introduce the following definition.

Definition 3.1 *Let $\gamma(C, p)$ be the maximum number of arcs of a digraph H with p vertices such that $L(H, e) \leq C$, for every arc e of $\vec{C}_p$.*

The next lemma gives the value of $\gamma(C,p)$ and shows that, in fact, the maximum number of requests we can groom is attained by taking those of minimum length. It is worth to mention that this property is not true if the physical graph is a path, as shown with a counterexample in [43].

Proposition 3.2 *Let $C = \frac{k(k+1)}{2} + r$, with $0 \le r \le k$. Then*

$$\gamma(C,p) = \begin{cases} \frac{p(p-1)}{2} & , \text{ if } p \le 2k+1, \text{ or } p = 2k+2 \text{ and } r \ge \frac{k+2}{2} \\ kp + 2r - 1 & , \text{ if } p = 2k+2 \text{ and } 1 \le r < \frac{k+2}{2} \\ kp + \left\lfloor \frac{rp}{k+1} \right\rfloor & , \text{ otherwise} \end{cases}$$

The graphs achieving $\gamma(C,p)$ are either the tournament T_p if p is small (namely, if $p \le 2k+1$ or $p = 2k+2$ and $r \ge \frac{k+2}{2}$), or subgraphs of a circulant digraph containing all the arcs of length $1,2,\ldots,k$, plus some arcs of length $k+1$ if $r > 0$.

Proof: We distinguish three cases according to the value of p.

Case 1. If p is small, that is such that the tournament T_p loads each arc at most C times, then $\gamma(C,p) = \frac{p(p-1)}{2}$. Let us now see for which values of p this fact holds.

If p is odd, the load of T_p is $\frac{p^2-1}{8} \le C$. The inequality $p^2-1 \le 8C$ implies $p^2-1 \le 4k(k+1)+8r$, and is satisfied if $p \le 2k+1$, as $p^2 - 1 \le 4k(k+1)$.

If p is even, the load of T_p is $\frac{p^2}{8} + \frac{1+\delta}{2}$, where $\delta = 1$ if $p \equiv 0 \pmod 4$ (see proof of Proposition 3.1).

If $p \le 2k$, it holds $\frac{p^2+8}{8} \le \frac{4k^2+8}{8} \le \frac{k(k+1)}{2} \le C$.

For $p = 2k+2$, it holds $\frac{p^2}{8} + \frac{1+\delta}{2} = \frac{k^2}{2} + k + 1 + \frac{\delta}{2} \le \frac{k^2+k}{2} + r = C$ if and only if $r \ge \frac{k+2+\delta}{2}$, with $\delta = 1$ if $p \equiv 0 \pmod 4$, that is, if k is odd. Therefore, the condition is satisfied if $r \ge \frac{k+2}{2}$.

In the next two cases, we provide first a lower bound on $\gamma(C,p)$, and then we prove a matching upper bound.

Case 2. If $p = 2k+2$ and $1 \le r < \frac{k+2}{2}$, a solution is obtained by taking all the arcs of length $1,2,\ldots,k\left(= \frac{p-2}{2}\right)$ – giving a load of $\frac{k(k+1)}{2}$ – plus $2r-1$ arcs of length $\frac{p}{2}$. For example, we can take the arcs $\left(i, i+\frac{p}{2}\right)$ for $i = 0,2,\ldots,2r-2\left(< \frac{p}{2}\right)$ and the arcs $\left(i, i-\frac{p}{2}\right)$ for $i = 1,3,\ldots,2r-3$. The load due to these arcs is at most r. Therefore, in this case $\gamma(C,p) \ge kp + 2r - 1$.

Case 3. If $p > 2k+2$ or $p = 2k+2$ and $r = 0$, a solution is obtained by taking all the arcs of length $1,2,\ldots,k$ plus $\left\lfloor \frac{rp}{k+1} \right\rfloor$ arcs of length $k+1$, in such a way that the load due to these arcs is at most C, which is always possible (for example, if p is prime with $k+1$, we take the requests $((k+1)i,(k+1)(i+1))$ for $0 \le i \le \left\lfloor \frac{rp}{k+1} \right\rfloor - 1$, the indices being taken modulo p). Therefore, in this case

$$\gamma(C,p) \ge kp + \left\lfloor \frac{rp}{k+1} \right\rfloor. \tag{3.4}$$

Let us now turn to upper bounds. Suppose we have a solution with γ arcs, γ_i being of length i on $\vec{C}_p$. As each arc of length i loads i arcs, and the total load of the arcs of $\vec{C}_p$ is

at most Cp, we have that

$$\begin{aligned} Cp &\geq \sum_{i=1}^{\infty} i\gamma_i \geq \sum_{i=1}^{k} i\gamma_i + (k+1)\left(\gamma - \sum_{i=1}^{k}\gamma_i\right) \\ &= \sum_{i=1}^{k} ip + (k+1)(\gamma - kp) + \sum_{i=1}^{k} \underbrace{(k+1-i)(p-\gamma_i)}_{\geq 0} \\ &\geq \frac{k(k+1)}{2}\cdot p + (k+1)(\gamma - kp). \end{aligned}$$

Since $Cp = \frac{k(k+1)}{2}\cdot p + rp$, we obtain $rp \geq (k+1)(\gamma - kp)$, and therefore

$$\gamma(C,p) \leq kp + \frac{rp}{k+1}\ . \tag{3.5}$$

Combining Equations (3.4) and (3.5), we get the result for case 3. For case 2, i.e., when $p = 2k+2$ and $1 \leq r < \frac{k+2}{2}$, Equation (3.5) yields $\gamma(C,p) \leq kp + 2r$. If we have equality, then necessarily $\gamma_i = p$ for $i = 1,\ldots,k$, so we have all arcs of length at most k. However, the $2r$ arcs of length at least $k+1$ induce a load at least $r+1$ on some arc of $\vec{C}_p$, so the total load would be strictly greater than C. Therefore, we have at most $\gamma(C,p) \leq kp + 2r - 1$, which gives the result. □

We define the parameter $\rho(C) = \max_p \left\{\frac{\gamma(C,p)}{p}\right\}$.

Proposition 3.3 *Let $C = k(k+1)/2 + r$, with $0 \leq r \leq k$. Then $\rho(C) \leq k + r/k + 1$.*

Proof: In Case 1 of the proof of Proposition 3.2, $\rho(C) \leq \frac{p-1}{2}$. If $p \leq 2k+1$, $\rho(C) \leq k$. If $p = 2k+2$ and $r \geq \frac{k+2}{2}$, $\rho(C) = k + \frac{1}{2} < k + \frac{r}{k+1}$. Otherwise, by Equation (3.5),

$$\rho(C) \leq \frac{kp + \frac{rp}{k+1}}{p} = k + \frac{r}{k+1}, \tag{3.6}$$

where $C = \frac{k(k+1)}{2} + r$, with $0 \leq r \leq k$. So, in all cases, $\rho(C) \leq k + \frac{r}{k+1}$. □

Table 3.1 shows the parameter $\gamma(C,p)$ for small values of C and p, as well as the parameter $\rho(C)$.

3.2.3 General lower bounds

By Propositions 3.1 and 3.2, Equations (3.1), (3.2), and (3.3) become

$$A = \sum_{p=2}^{n} pa_p \tag{3.7}$$

$$\sum_{p=2}^{n} a_p \geq \left\lceil \frac{n^2+\alpha}{8C} \right\rceil, \text{ where } \alpha = \begin{cases} -1 & \text{, if } n \text{ is odd} \\ 4 & \text{, if } n \equiv 2 \pmod 4 \\ 8 & \text{, if } n \equiv 0 \pmod 4 \end{cases} \tag{3.8}$$

$$\sum_{p=2}^{n} a_p\gamma(C,p) \geq \frac{n(n-1)}{2} \tag{3.9}$$

p	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	$\rho(C)$
$C = 1$	1	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	1
$C = 2$	1	3	5	7	9	10	**12**	13	**15**	16	**18**	19	**21**	22	**24**	3/2
$C = 3$	1	3	6	**10**	**12**	**14**	**16**	**18**	**20**	**22**	**24**	**26**	**28**	**30**	**32**	2
$C = 4$	1	3	6	10	13	16	18	**21**	23	25	**28**	30	32	**35**	37	7/3
$C = 5$	1	3	6	10	15	18	21	**24**	26	29	**32**	34	37	**40**	42	8/3
$C = 6$	1	3	6	10	15	**21**	**24**	**27**	**30**	**33**	**36**	**39**	**42**	**45**	**48**	3
$C = 7$	1	3	6	10	15	21	25	29	32	35	**39**	42	45	48	**52**	13/4
$C = 8$	1	3	6	10	15	21	27	31	35	38	**42**	45	49	52	**56**	14/4
$C = 9$	1	3	6	10	15	21	28	33	37	41	**45**	48	52	56	**60**	15/4
$C = 10$	1	3	6	10	15	21	28	**36**	**40**	**44**	**48**	**52**	**56**	**60**	**64**	4

Table 3.1: The parameter $\gamma(C, p)$ for some values of C and p, as well as $\rho(C)$. The **bold** values achieve $\rho(C)$.

We are ready to prove the general lower bound on the number of ADMs used by any solution.

Theorem 3.1 (General Lower Bound) *Let $C = \frac{k(k+1)}{2} + r$, with $0 \le r \le k$. The number of ADMs required in a bidirectional ring with n nodes and grooming factor C satisfies*

$$A(C,n) \;\ge\; \left\lceil \frac{n(n-1)}{2 \cdot \rho(C)} \right\rceil \;=\; \left\lceil \frac{n(n-1)}{2} \frac{k+1}{k(k+1)+r} \right\rceil. \tag{3.10}$$

Proof: Using Equations (3.7) and (3.9), and the definition of $\rho(C)$, we get that the number A of ADMs used by any solution satisfies

$$\frac{n(n-1)}{2} \;\le\; \sum_{p=2}^{n} a_p \cdot \gamma(C,p) \;=\; \sum_{p=2}^{n} p \cdot a_p \cdot \rho(C) \;=\; \rho(C) \cdot A.$$

From the above equation and using Equation (3.6), we get

$$A \;\ge\; \left\lceil \frac{n(n-1)}{2 \cdot \rho(C)} \right\rceil \;=\; \left\lceil \frac{n(n-1)}{2} \frac{k+1}{k(k+1)+r} \right\rceil.$$

□

To achieve the lower bound of Theorem 3.1, the only possibility is to use graphs on p vertices with $\gamma(C,p)$ arcs. The **bold** values in Table 3.1 achieve $\rho(C)$, and therefore the subgraphs corresponding to those values (which exist by Proposition 3.2) are good candidates to construct an optimal partition of the request graph.

Comparison of existing lower bounds. In [72] the Ring Traffic Grooming problem in the bidirectional ring is studied. The authors state a lower bound regardless of routing for a general set of requests. In the particular case of uniform traffic, they get a

lower bound of $\frac{n^2-1}{4\sqrt{2C}}$ (see [72, Theorem 1, page 198]). They indicate in their article that they can improve this bound by a factor of 2 for all-to-all uniform unitary traffic. We thank T. Chow and P. Lin for sending us the proof of the following theorem, which is only announced in [72]:

Theorem 3.2 ([71, 72]) *If a traffic instance of ring grooming is uniform and unitary, then, regardless of routing,*

$$A(C,n) \geq \frac{1}{2\sqrt{C}} \sqrt{\frac{n^2(n-1)^2}{2} - n(n-1)}.$$

The lower bound we obtained in Theorem 3.1 is greater than the bound of Theorem 3.2, but it should be observed that we restrict ourselves to shortest path symmetric routing. Our bound is $\frac{n(n-1)}{2\rho(C)}$ and the lower bound of Theorem 3.2 is less than $\frac{n(n-1)}{2\sqrt{2C}}$. The fact that our bound is better follows from the fact that $\rho(C) < \sqrt{2C}$. Indeed,

$$\rho^2(C) \leq \left(k + \frac{r}{k+1}\right)^2 = k^2 + \frac{2kr}{k+1} + \frac{r^2}{(k+1)^2} < k^2 + 2r + 1 < k^2 + k + 2r = 2C.$$

3.3 Case $C = 1$

For $C = 1$, by Proposition 3.2 $\gamma(1,p) = p$ if $p \geq 2$. Furthermore, all the directed cycles achieve $\rho(1)$ (see Table 3.1).

Theorem 3.3

$$A(1,n) = \begin{cases} \frac{n(n-1)}{2} & , \text{ if } n \text{ is odd} \\ \frac{n^2}{2} & , \text{ if } n \text{ is even} \end{cases}$$

Proof: For $C = 1$, the only possible subgraphs involved in the partition of the edges of T_n are cycles and paths. If only cycles are used, the total number of ADMs is $\frac{n(n-1)}{2}$, which equals the lower bound of Theorem 3.1. Each path involved in the partition adds one unity of cost with respect to $\frac{n(n-1)}{2}$.

If $n = 2q+1$ is odd, by [51, Theorem 3.3] we know that the arcs of T_n can be covered with q $\vec{C}_3$'s and $\frac{q(q-1)}{2}$ $\vec{C}_4$'s. The total number of vertices of this construction is $3q + 2q(q-1) = q(2q+1) = \frac{n(n-1)}{2}$.

If n is even, each vertex must appear with odd degree in at least one subgraph, so the number of paths in any construction is at least $n/2$. Therefore, the lower bound becomes $\frac{n(n-1)}{2} + \frac{n}{2} = \frac{n^2}{2}$. By [51, Theorem 3.4] the arcs of T_n can be covered with

- 4 $\vec{C}_3$'s and $2q^2 - 3$ $\vec{C}_4$'s, if $n = 4q$ with $q > 1$;
- 2 $\vec{C}_3$'s and $2q^2 + 2q - 1$ $\vec{C}_4$'s, if $n = 4q + 2$.

For $n = 4$, we cover T_4 with a $\vec{C}_4$ and two arcs. Note that in these constructions, some arcs are covered more than once. In both cases, the total number of vertices of the construction is $\frac{n^2}{2}$, hence the lower bound is attained.

Finally, one can check that in the constructions of [51], the length of the arcs involved in the covering of T_n is in all cases bounded above by $\left\lfloor \frac{n}{2} \right\rfloor$, and therefore all the cycles induce load 1. □

Remark 3.2 *For the original problem with $G = C_n^*$ and $I = K_n^*$, if we apply Theorem 3.3 we get in the case $n/2$ a value of n^2 ADMS; but if we delete the constraint of symmetric routings we get a value of $n(n-1)/2$ by using [51, Theorems 4.1 and 4.2] (however these constructions use many K_2's).*

3.4 Case $C = 2$

When $C = 2$ the general lower bound of Theorem 3.1 gives $A(2, n) \geq \frac{n(n-1)}{3}$. We first improve this bound in Section 3.4.1, and then give solutions with a good approximation ratio in Section 3.4.2.

3.4.1 Improved lower bounds

For $C = 2$, by Proposition 3.2 $\gamma(2, 2) = 1$, $\gamma(2, 3) = 3$, $\gamma(2, 4) = 5$ (note that $\gamma(2, 4) = 6$ if the routing is not restricted to be symmetric), and $\gamma(2, p) = \left\lfloor \frac{3p}{2} \right\rfloor$ for $p \geq 5$. The optimal solutions for $p \geq 4$ even consist of the p arcs of length 1 $(i, i + 1)$ for $0 \leq i \leq p - 1$, plus the $p/2$ arcs of length 2 $(2i, 2i + 2)$ for $0 \leq i \leq p/2 - 1$ (in fact, triangles sharing a vertex; see Figure 3.3 for $p = 6$). For p odd we have two classes of optimal graphs (see Figure 3.3 for $p = 5$).

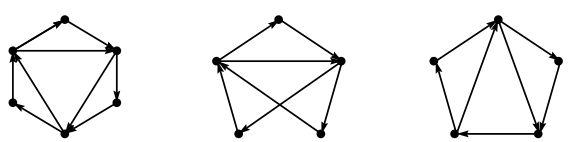

Figure 3.3: Some admissible digraphs for $C = 2$.

Equation (3.9) becomes in the case $C = 2$

$$\sum_{p=2}^{n} a_p \gamma(2, p) = a_2 + 3a_3 + 5a_4 + 7a_5 + 9a_6 + 10a_7 + 12a_8 + \ldots \geq \frac{n(n-1)}{2}.$$

Therefore,

$$A = \sum_{p=2}^{n} p a_p \geq \frac{2}{3} \sum_{p=2}^{n} a_p \gamma(2, p) + \frac{4}{3} a_2 + a_3 + \frac{2}{3} a_4 + \frac{1}{3}(a_5 + a_7 + a_9 + \ldots) \quad (3.11)$$

$$\geq \frac{n(n-1)}{3} + \frac{4}{3} a_2 + a_3 + \frac{2}{3} a_4 + \frac{1}{3}(a_5 + a_7 + a_9 + \ldots). \quad (3.12)$$

We can already see that the bound $\frac{n(n-1)}{3}$ cannot be attained. Indeed, to reach it we need to use only graphs with $6, 8, 10, \ldots$ vertices. But the number of graphs Λ satisfies, by Proposition 3.1, $\Lambda \geq \frac{n^2-1}{16}$, so $A \geq 6\frac{n^2-1}{16} > \frac{n(n-1)}{3}$.

The following proposition gives a lower bound of order $\frac{11}{32}n(n-1)$. Note that $11/32 > 11/33 = 1/3$.

Proposition 3.4 (Tighter Lower Bound for $C = 2$)

$$A(2,n) \geq \left\lceil \frac{11n^2 - 8n - 3}{32} \right\rceil = \left\lceil \frac{11}{16}\frac{n(n-1)}{2} + \frac{3n-3}{32} \right\rceil. \tag{3.13}$$

Proof: We can write $A \geq 6(\Lambda - a_2 - a_3 - a_4 - a_5) + 2a_2 + 3a_3 + 4a_4 + 5a_5$, that is,

$$A \geq 6\Lambda - (4a_2 + 3a_3 + 2a_4 + a_5). \tag{3.14}$$

From Equations (3.11) and (3.12) we get that

$$3A \geq n(n-1) + (4a_2 + 3a_3 + 2a_4 + a_5). \tag{3.15}$$

Summing Equations (3.14) and (3.15) gives

$$4A \geq 6\Lambda + n(n-1). \tag{3.16}$$

By Proposition 3.1, we have that

$$\Lambda \geq \frac{n(n-1)}{16} + \frac{n+\alpha}{16}. \tag{3.17}$$

Combining Equations (3.16) and (3.17) and using that $\alpha \geq -1$ yields

$$A \geq \frac{11n(n-1)}{32} + \frac{3n}{32} + \frac{3\alpha}{32} \geq \frac{11n^2 - 8n - 3}{32}.$$

□

3.4.2 Upper bounds

In this section we build families of solutions for $C = 2$. We conjecture that there exists a decomposition using A vertices with ratio $\frac{A}{\frac{n(n-1)}{2}}$ of order $\frac{11}{16}$, which would be optimal by Proposition 3.4. For that, we should find some (multipartite) graphs achieving this ratio. A candidate is $K_{4,4,4}$, which has 48 edges. Unfortunately, we have not been able to cover it with 33 vertices (which would achieve the optimal ratio) but only with 34, giving a 34/33-approximation.

For the sake of the presentation, we first present a simple 12/11-approximation inspired from a construction of [51].

A 12/11-approximation

This construction is defined recursively. Suppose we have a solution for n vertices using A_n ADMs, with $n = 2p$ or $n = 2p+1$. Let the vertex set be labeled $0_A < 1_A < \ldots < (p-1)_A < 0_B < 1_B < \ldots < (p-1)_B$, plus ∞ is n is odd. For $n+2$, we add two vertices x_A and x_B with the order $x_A < 0_A < 1_A < \ldots < (p-1)_A < x_B < 0_B < 1_B < \ldots < (p-1)_B < \infty$. We use as subdigraphs those of the solution for n plus the $\lfloor p/2 \rfloor$ digraphs on the 6 vertices $x_A, i_A, (i+\lfloor p/2 \rfloor)_A, x_B, i_B, (i+\lfloor p/2 \rfloor)_B$ and the 8 arcs (x_A, i_A), $(x_A, (i+\lfloor p/2 \rfloor)_A)$, (i_A, x_B), $((i+\lfloor p/2 \rfloor)_A, x_B)$, (x_B, i_B), $(x_B, (i+\lfloor p/2 \rfloor_B)$, (i_B, x_A), $((i+\lfloor p/2 \rfloor)_B, x_A)$, for $0 \leq i \leq \lfloor p/2 \rfloor - 1$.

If $n = 2p$ with p even, there remains uncovered the arc (x_A, x_B).

If $n = 2p+1$ with p even, there remain the 3 arcs $(x_A, x_B), (x_B, \infty)$, and (∞, x_A), which we cover with the circuit (x_A, x_B, ∞).

If $n = 2p$ with p odd, there remain the 5 arcs $(x_A, (p-1)_A), ((p-1)_A, x_B), (x_B, (p-1)_B), ((p-1)_B, x_A)$, and (x_A, x_B), which we cover with a digraph on 4 vertices containing all of them.

Finally, if $n = 2p+1$ with p odd, there remain the 7 arcs $(x_A, (p-1)_A), ((p-1)_A, x_B), (x_B, (p-1)_B), ((p-1)_B, x_A)$, $(x_A, x_B), (x_B, \infty)$, and (∞, x_A), which we cover with a digraph on 5 vertices containing all of them.

One can check that, in all cases, the arcs (u, v) considered satisfy $d_{\vec{C}_n}(u, v) \leq n/2$.

To compute the number of ADMs of this construction, we have the recurrence relations $A_{4q+2} = A_{4q} + 6q + 2$, $A_{4q+4} = A_{4q+2} + 6q + 4$, $A_{4q+3} = A_{4q+1} + 6q + 3$, and $A_{4q+5} = A_{4q+3} + 6q + 5$. Starting with $A_2 = 2$ or $A_4 = 6$ (obtained with the partition with the digraph on 4 vertices formed by the C_4 $(0, 1, 2, 3)$ plus the arc $(0, 2)$ and the digraph on 2 vertices $(1, 3)$) and $A_3 = 3$ or $A_5 = 8$ (obtained with the partition of T_5 using the first digraph on 5 vertices of Figure 3.3 and the remaining T_3), we get $A_{4q} = 6q^2 = \frac{6n^2}{16}$, $A_{4q+2} = 6q^2 + 6q + 2 = \frac{6n^2+8}{16}$, $A_{4q+1} = 6q^2 + 2q = \frac{6n^2-4n-2}{16}$, and $A_{4q+3} = 6q^2 + 8q + 3 = \frac{6n^2-4n+6}{16}$.

In all cases, the number of ADMs is of order $\frac{6}{8}\frac{n(n-1)}{2}$, so asymptotically the ratio between the number of ADMs of this construction and the lower bound of Proposition 3.4 tends to $\frac{6}{8}\frac{16}{11} = \frac{12}{11}$.

A 34/33-approximation

It will be useful to use the notation G_5 and G_6 to refer to the digraphs depicted in Figure 3.4. The key idea of this construction is that an oriented tripartite graph $K_{4,4,4}$ can be partitioned into admissible subdigraphs for $C = 2$ using 34 vertices overall, as follows.

Let the tripartition classes of the $K_{4,4,4}$ be $\{1_A, 1_B, 1_C, 1_D\}$, $\{2_A, 2_B, 2_C, 2_D\}$, $\{3_A, 3_B, 3_C, 3_D\}$, and let the vertices be ordered in the ring $1_A < 2_A < 3_A < 1_B < 2_B < 3_B < 1_C < 2_C < 3_C < 1_D < 2_D < 3_D$. The arcs of an oriented $K_{4,4,4}$ can be partitioned into 4 G_6's with $\{x_1, x_2, x_3, x_4, x_5, x_6\} = \{1_A, 2_A, 3_B, 1_C, 2_C, 3_D\}$, $\{1_B, 2_B, 3_B, 1_D, 2_D, 3_D\}$, $\{1_B, 2_C, 3_C, 1_D, 2_A, 3_A\}$, and $\{1_A, 3_A, 2_B, 1_C, 3_C, 2_D\}$, plus 2 G_5's with $\{x_1, x_2, x_3, x_4, x_5\} = \{3_A, 1_C, 2_C, 1_D, 2_D\}$ and $\{3_D, 2_A, 2_B, 1_D, 1_C\}$ (see Figure 3.4). The total number of vertices of this partition is 34.

We are now ready to explain the construction. We take an integer $p \equiv 1$ or $3 \pmod 6$, hence K_p can be partitioned into triangles. We replace each vertex i of K_p with 4 vertices

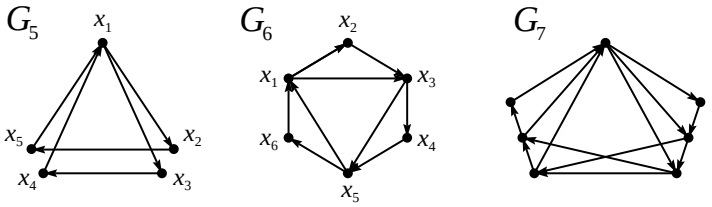

Figure 3.4: Digraphs G_5 and G_6 used in the 34/33-approximation for $C = 2$, and digraph G_7 suitable for $C = 3$ referred in the proof of Proposition 3.6.

i_A, i_B, i_C, i_D, and order the vertices $1_A < \ldots < p_A < 1_B < 2_B < \ldots < p_B < 1_C < \ldots < p_C < 1_D < \ldots < p_D$. To a triple $\{i, j, k\}$ corresponding to a triangle of K_p, with $i < j < k$, we associate the decomposition described above of the $K_{4,4,4}$ on vertices $\{\ell_A, \ell_B, \ell_C, \ell_D : \ell = i, j, k\}$. In this way, $K_{p\times 4}$ can be partitioned into $\frac{p(p-1)}{6}$ $K_{4,4,4}$'s, or equivalently into $\frac{p(p-1)}{6} \cdot 4$ G_6's and $\frac{p(p-1)}{6} \cdot 2$ G_5's. Overall, we use $\frac{34p(p-1)}{6}$ vertices. Each of the subdigraphs of this partition is admissible, as the distance in the ring between the endpoints of an arc is strictly smaller than $2p$.

To partition an oriented K_{4p}, there remain only the K_4's induced inside each class of the $K_{p\times 4}$. As $A(2, 4) = 6$, we use $6p$ vertices to cover all the K_4's.

Therefore, if $p \equiv 1$ or $3 \pmod 6$, an oriented K_{4p} can be partitioned using $6p + \frac{34p(p-1)}{6} = \frac{34p^2+2p}{6} = \frac{34n^2+8n}{96}$ vertices. To decompose K_{4p+1}, we add a vertex ∞, and we partition the p K_5's using 8 vertices for each one of them. Overall, we use $8p + \frac{34p(p-1)}{6} = \frac{34p^2+14p}{6} = \frac{34n^2-12n-24}{96}$ vertices.

If $p \not\equiv 1$ or $3 \pmod 6$, we introduce dummy vertices to get $p' \equiv 1$ or $3 \pmod 6$, we do the construction described above, and then we remove the dummy edges and vertices. It is clear that these dummy vertices add $O(n)$ vertices to the construction, hence the coefficient of the term n^2 remains the same.

Since $\frac{33n^2-24n-9}{96}$ is a lower bound by Proposition 3.4, this construction constitutes a 34/33-approximation.

3.5 Case $C = 3$

We first provide improved lower bounds for some congruence classes in Section 3.5.1 and then we provide constructions in Section 3.5.2, which are either optimal or asymptotically optimal.

3.5.1 Improved lower bounds

In this case (see Table 3.1) we have $\gamma(3, 2) = 1$, $\gamma(3, 3) = 3$, $\gamma(3, 4) = 6$, and $\gamma(3, p) = 2p$ for $p \geq 5$, so $\rho(3) = 2$. Therefore, by Theorem 3.1, we get

Proposition 3.5 $A(3,n) \geq \frac{n(n-1)}{4}$.

By Equations (3.7) and (3.9) we have

$$\begin{aligned} 2A &= \sum_{p=2}^{n} 2pa_p = 4a_2 + 6a_3 + 8a_4 + \sum_{p=5}^{n} 2pa_p \\ \frac{n(n-1)}{2} &\leq \sum_{p=2}^{n} a_p\gamma(3,p) = a_2 + 3a_3 + 6a_4 + \sum_{p=5}^{n} 2pa_p \end{aligned}$$

So,

$$A \geq \frac{n(n-1)}{4} + \frac{3}{2}a_2 + \frac{3}{2}a_3 + a_4.$$

Therefore, if the lower bound is attained, then necessarily $a_2 = a_3 = a_4 = 0$. We will see in Section 3.5.2 that this is the case for $n \equiv 1$ or $5 \pmod{12}$, using optimal digraphs on 5 vertices (namely T_5) and on 6 vertices (namely $\vec{K}_{2,2,2}$, see Figure 3.5). Optimal graphs are obtained by using arcs of length 1 and 2, so the degree of any vertex in an optimal subdigraph is 4. That is possible only if the total degree of a vertex, namely $n-1$, is a multiple of 4. Otherwise, the following proposition shows that the lower bound of Proposition 3.5 cannot be attained.

Proposition 3.6
If $n \equiv 3 \pmod 4$, $A(3,n) \geq \frac{n(n-1)}{4} + \frac{n}{6} = \frac{3n^2-n}{12}$.
If $n \equiv 0 \pmod 2$, $A(3,n) \geq \frac{n(n-1)}{4} + \frac{n}{4} = \frac{n^2}{4}$.

Proof: We use the following observation: If a vertex x has out-degree 3 (resp. in-degree 3) in a digraph B_λ, then its nearest out-neighbor A_x^+ (resp. in-neighbor A_x^-) has in-degree 1 and out-degree at most 1 (resp. out-degree 1 and in-degree at most 1). Indeed, suppose x has out-degree 3, and let A_x^+, B_x^+, C_x^+ be the out-neighbors of x. Then the load of the arc entering A_x^+ is already 3, so A_x^+ has no other in-neighbor than x. The load of the arc leaving A_x^+ is already 2, so A_x^+ has at most 1 out-neighbor y. If y has 2 or more in-neighbors, then A_x^+ is not its nearest one. Hence, to each vertex x of out-degree 3 (resp. in-degree 3) is associated a distinct vertex A_x^+ (resp. A_x^-) of degree at most 2.

Consider the digraphs in which a given vertex x appears. Let α_i^x be the number of times x appears with degree i, and let $\alpha_i = \sum_x \alpha_i^x$. Vertex x appears in $\sum_i \alpha_i^x$ digraphs, so

$$A = \sum_x \sum_i \alpha_i^x = \sum_i \alpha_i. \tag{3.18}$$

As each vertex has degree $n-1$, $n-1 = \sum_i i \cdot \alpha_i^x$, and so

$$n(n-1) = \sum_x \sum_i i \cdot \alpha_i^x = \sum_i i \cdot \alpha_i. \tag{3.19}$$

Due to the load constraint, a vertex has out-degree (resp. in-degree) at most 3 in all the digraphs in which it appears. Therefore, its degree is at most 6, that is, $\alpha_i = 0$ for $i \geq 7$. Furthermore, by the above observation if a vertex has degree 6 (resp. 5), to this vertex

are associated 2 vertices (resp. 1 vertex) of degree at most 2, and all these vertices are distinct, so

$$\alpha_1 + \alpha_2 \geq 2\alpha_6 + \alpha_5. \tag{3.20}$$

Combining Equations (3.18) and (3.19) we get

$$4A = n(n-1) + 3\alpha_1 + 2\alpha_2 + \alpha_3 - \alpha_5 - 2\alpha_6. \tag{3.21}$$

We distinguish two cases: n even or $n = 4t + 3$.

If n is even, $n - 1$ is odd and each vertex must appear at least in one B_λ with odd degree, so

$$\alpha_1 + \alpha_3 + \alpha_5 \geq n. \tag{3.22}$$

Using Equation (3.20) multiplied by 2 in Equation (3.21) we get $4A \geq n(n-1) + \alpha_1 + \alpha_3 + \alpha_5 + 2\alpha_6$, so by Equation (3.22), $4A \geq n(n-1) + n$, as claimed. Note that to obtain equality we need $\alpha_6 = 0$, $\alpha_1 + \alpha_2 = \alpha_5$, and $\alpha_1 + \alpha_3 + \alpha_5 = n$.

If $n = 4t + 3$, the degree of each vertex satisfies $n - 1 \equiv 2 \pmod 4$, so no vertex can appear with degree 4 in all the digraphs. Each vertex must appear either at least once with degree 6 or 2, or at least twice with odd degree (for example, 5 and 5, 3 and 3, 1 and 1, or 5 and 1), so

$$\alpha_2 + \alpha_6 + \frac{1}{2}(\alpha_1 + \alpha_3 + \alpha_5) \geq n. \tag{3.23}$$

Equation (3.21) can be rewritten as

$$4A = n(n-1) + \frac{2}{3}\left(\alpha_2 + \alpha_6 + \frac{1}{2}(\alpha_1 + \alpha_3 + \alpha_5)\right) + \frac{4}{3}(\alpha_2 + \alpha_1 - 2\alpha_6 - \alpha_5) + \frac{2}{3}\alpha_3 + \frac{4}{3}\alpha_1. \tag{3.24}$$

Using Equations (3.20) and (3.23) in Equation (3.24) yields $4A \geq n(n-1) + \frac{2}{3}n + \frac{2}{3}\alpha_3 + \frac{4}{3}\alpha_1$, or $A \geq \frac{n(n-1)}{4} + \frac{n}{6}$, as claimed. Note that to reach the equality, we need to have $\alpha_1 = \alpha_3 = 0$, $\alpha_2 = 2\alpha_6 + \alpha_5$ by Equation (3.20), and $2\alpha_6 + 2\alpha_2 + \alpha_5 = 2n$ by Equation (3.23), so $\alpha_2 = \frac{2n}{3}$, hence an optimal decomposition should use $\frac{n}{3}$ digraphs like the digraph G_7 depicted in Figure 3.4, having 1 vertex of degree 6 and 2 vertices of degree 2. □

3.5.2 Constructions

Our constructions rely on the existence of 3-*GDD*'s, that is, decompositions of complete multipartite graphs into K_3's. We recall the definition and some basic results below.

Decompositions or complete multipartite graphs into K_3's. Let $v_1, v_2, \ldots, v_q$ be non-negative integers; the complete multipartite graph with group sizes $v_1, v_2, \ldots, v_q$ is defined to be the graph with vertex set $V_1 \cup V_2 \cup \cdots \cup V_q$ where $|V_i| = v_i$, and two vertices $u \in V_i$ and $v \in V_j$ are adjacent if $i \neq j$. Using terminology of design theory, the graph of type $p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_h^{\alpha_h}$ is the complete multipartite graph with α_i groups of size p_i. The existence of a partition of this multipartite graph into K_k's is equivalent to the existence of a *k-GDD* (*Group Divisible Design*) of type $p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_h^{\alpha_h}$ (see [75]). Here we are interested in the existence of 3-*GDD*'s, that is, partitions into K_3's. When $|V_i| = p$ for all i, we denote by $K_{p\times q}$ the multipartite graph of type p^q. Trivial necessary conditions for the existence of a 3-*GDD* are

(i) the degree of each vertex is even; and

(ii) the number of edges is a multiple of 3.

These conditions are in general sufficient. In particular, the following results will be used later.

Theorem 3.4 ([75])
A 3-GDD of type 2^q *with* $q \geq 3$ *exists if and only if* $q \equiv 0$ *or* $1 \pmod 3$.
A 3-GDD of type $2^{q-1}4$ *with* $q \geq 4$ *exists if and only if* $q \equiv 1 \pmod 3$.
A 3-GDD of type 3^q *with* $q \geq 3$ *exists if and only if* q *is odd.*
A 3-GDD of type $3^{q-1}1$ *with* $q \geq 3$ *exists if and only if* q *is odd.*
A 3-GDD of type $3^{q-1}5$ *with* $q \geq 5$ *exists if and only if* q *is odd.*
A 3-GDD of type $3^{q-1}11$ *with* $q \geq 7$ *exists if and only if* q *is odd.*

The basic partition. In what follows $\vec{K}_{2,2,2}$ will denote the digraph on 6 vertices and 12 arcs depicted in Figure 3.5. This digraph can be viewed as being obtained from the K_3 (i, j, k) with $i < j < k$ by replacing each vertex i with two vertices i_A and i_B forming an independent set.

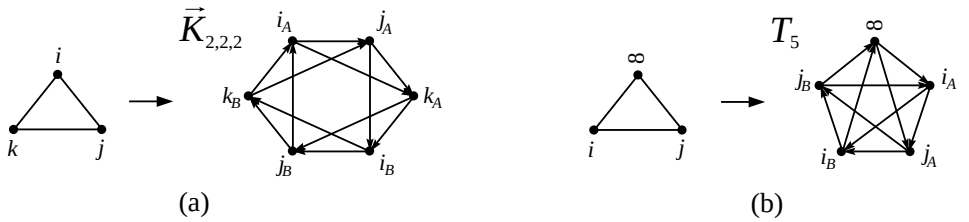

Figure 3.5: (a) Digraph $\vec{K}_{2,2,2}$ obtained from K_3 (i, j, k), with $i < j < k$; (b) digraph T_5 obtained from a K_3 of the form (∞, i, j).

Note that $\vec{K}_{2,2,2}$ is an optimal digraph for $C = 3$, since it attains the ratio $\rho(3) = 2$ (see Table 3.1). The idea of the constructions consists in starting from some graph G (mainly a multipartite graph) which can be decomposed into K_3's, replacing each vertex with two non-adjacent vertices, and then using the following lemma.

Lemma 3.1 *If a graph* $G = (V, E)$ *with vertex set* $\{1, 2, \ldots, |V|\}$ *can be decomposed into* h K_3*'s, then the digraph* H *obtained from* G *by replacing each vertex* i *with two non-adjacent vertices* i_A *and* i_B*, and where the vertices are ordered* $1_A, 2_A, \ldots, |V|_A, 1_B, 2_B, \ldots, |V|_B$*, has a valid decomposition into* $\vec{K}_{2,2,2}$*'s with a total of* $6h$ *vertices.*

Proof: To each triangle (i, j, k) with $1 \leq i < j < k \leq |V|$ is associated the $\vec{K}_{2,2,2}$ with vertices $1 \leq i_A < j_A < k_A \leq |V| < i_B < j_B < k_B \leq 2|V|$. To show that the decomposition is valid for $C = 3$, it suffices to show that the distance between the end-vertices of any arc of any $\vec{K}_{2,2,2}$ is at most $|V|$. That is true for the arcs (x_A, y_A) or (x_B, y_B) as they satisfy $x < y$, and also for the arcs (x_A, y_B) or (x_B, y_A) as they satisfy $x > y$ (see Figure 3.5(a)). □

Some small cases. We provide here decompositions of some particular small digraphs that will be used in the constructions of Propositions 3.8 and 3.9.

Lemma 3.2 *$A(3,5) = 5$, $A(3,6) \leq 10$, $A(3,7) \leq 12$, $A(3,8) \leq 18$, $A(3,9) \leq 21$, $A(3,10) \leq 28$, $A(3,11) \leq 31$, and $A(3,23) \leq 132$.*

Proof: Case $n = 5$. The decomposition is given in Figure 3.5(b), and can be viewed as obtained from the K_3 (∞, i, j) by replacing each of i, j with two vertices.

Case $n = 6, 7$. The complete graph K_4 can be decomposed into one $K_{1,3}$ $(0; \infty, 1, 2)$ and one K_3 $(\infty, 1, 2)$. Replace each of the vertices i, j, k with two vertices. The T_7 on the ordered vertices $\infty, 0_A, 1_A, 2_A, 0_B, 1_B, 2_B$ can be partitioned into a T_5 on $\infty, 1_A, 2_A, 1_B, 2_B$ ((see Figure 3.5(b) with $i = 1, j = 2$)) and the admissible digraph on 7 vertices and 11 arcs depicted in Figure 3.6(b) with $i = 0, j = 1, k = 2$. So we obtained a valid decomposition using 12 vertices. Deleting vertex ∞ yields a decomposition of T_5 with 10 vertices.

Case $n = 8, 9$. K_5 is the union of two K_3's $(\infty, 1, 3)$, $(0, 2, 3)$ and a C_4 $(\infty, 0, 1, 2)$. Replacing each vertex with two vertices we get a partition of the T_9 on the ordered vertices $\infty, 0_A, 1_A, 2_A, 3_A, 0_B, 1_B, 2_B, 3_B$. Namely, to the K_3 $(\infty, 1, 3)$ we associate a T_5 on $\infty, 1_A, 3_A, 1_B, 3_B$ (see Figure 3.5(b) with $i = 1, j = 3$). To the K_3 $(0, 2, 3)$ we associate a $\vec{K}_{2,2,2}$ on $0_A, 2_A, 3_A, 0_B, 2_B, 3_B$. To the C_4 $(\infty, 0, 1, 2)$ we associate the digraph on 7 vertices of Figure 3.6(a) with $i = 0, j = 1, k = 2$ and the triangle $(1_A, 2_A, 2_B)$. Therefore, $A(3,9) \leq 21$. Vertex 1_A appears in 3 digraphs, so $A(3,8) \leq 21 - 3 = 18$.

Case $n = 10, 11$. K_6 can be partitioned into 3 K_3's $(\infty, 1, 3), (\infty, 2, 4), (0, 1, 4)$, a star $K_{1,3}$ $(0; \infty, 2, 3)$, and a P_4 $[1, 2, 3, 4]$. Replacing each vertex with two vertices we get a partition of the T_{11} on the ordered vertices $\infty, 0_A, 1_A, 2_A, 3_A, 4_A, 0_B, 1_B, 2_B, 3_B, 4_B$ into 2 T_5's on $\infty, 1_A, 3_A, 1_B, 3_B$ and $\infty, 2_A, 4_A, 2_B, 4_B$, a $\vec{K}_{2,2,2}$ on $0_A, 1_A, 4_A, 0_B, 1_B, 4_B$, a digraph on 7 vertices and 11 arcs depicted in Figure 3.6(b) with $i = 0, j = 2, k = 3$, and an admissible digraph on 8 vertices with arcs $(1_A, 2_A), (2_A, 3_A), (3_A, 4_A), (1_B, 2_B), (2_B, 3_B), (3_B, 4_B)$, $(2_A, 1_B), (2_B, 1_A), (3_A, 2_B), (3_B, 2_A), (4_A, 3_B), (4_B, 3_A)$. Therefore, $A(3,11) \leq 31$, and as vertex ∞ appears in 3 subgraphs, we get $A(3,10) \leq 28$.

Case $n = 23$. We decompose K_{12} into 19 K_3's and 3 $K_{1,3}$'s, where vertex ∞ appears in 5 K_3's and in a star $(i; \infty, j, k)$, the two other stars being of the form $(i'; j'k', \ell')$ with $i' < j' < k' < \ell'$. We obtain a decomposition of T_{23} into 5 T_5's, 14 $\vec{K}_{2,2,2}$'s, 1 digraph of Figure 3.6(a), and 2 digraphs of Figure 3.6(c). Thus, $A(3,23) \leq 5 \cdot 5 + 14 \cdot 6 + 7 + 8 + 8 = 132$. □

Constructions. We begin with an optimal partition for $n \equiv 0, 1, 4,$ or $5 \pmod{12}$, and then we provide near-optimal constructions for the remaining values.

Proposition 3.7
If $n \equiv 0$ or $4 \pmod{12}$, $A(3,n) = \frac{n^2}{4}$.
If $n \equiv 1$ or $5 \pmod{12}$, $A(3,n) = \frac{n(n-1)}{4}$.

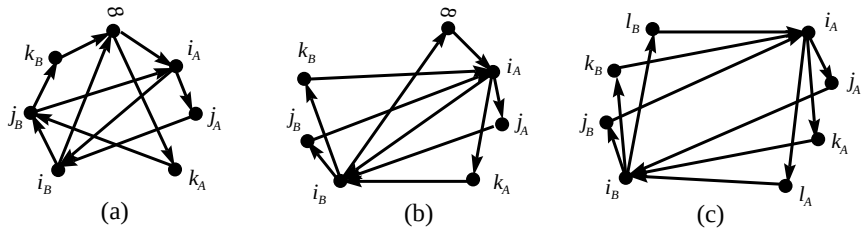

Figure 3.6: (a) Digraph associated to a C_4 (∞, i, j, k). Digraphs associated to stars ($K_{1,3}$'s), with $\infty < i < j < k < \ell$: (b) star of the form $(i; \infty, j, k)$; (c) star of the form $(i; j, k, \ell)$.

Proof: The lower bound follows from Propositions 3.5 and 3.6. For the upper bound, we will apply Lemma 3.1 with $G = K_{2\times q}$ (type 2^q), which can be decomposed by Theorem 3.4 into $\frac{2q(q-1)}{3}$ K_3's if $q \equiv 0$ or $1 \pmod 3$. As G has 2q vertices, the graph H described in Lemma 3.1 has $4q$ vertices and can be decomposed into admissible $\vec{K}_{2,2,2}$'s. Adding an admissible T_4 on each of the q independent sets of H (of the form $\{i_A, j_A, i_B, j_B\}$ where $\{i, j\}$ is an independent set of G), we get a valid decomposition of T_{4q} into q T_4's and $\frac{2q(q-1)}{3}$ admissible $\vec{K}_{2,2,2}$'s. So using $A(3,4) = 4$, we get $A(3,4q) \leq qA(3,4) + 4q(q-1) = 4q^2$ for $q \equiv 0$ or $1 \pmod 3$. So $A(3,n) \leq \frac{n^2}{4}$ for $n \equiv 0$ or $4 \pmod{12}$.

For $n = 4q+1$, we add to the vertex set of H an extra vertex ∞. Adding to the arcs of H the q tournaments T_5 built on $\infty, i_A, j_A, i_B, j_B$, where vertices i, j are not adjacent in G, we get a decomposition of T_{4q+1} into q admissible T_5's plus $\frac{2q(q-1)}{3}$ admissible $\vec{K}_{2,2,2}$'s (the distance being at most $2q-1$ in H and so $2q$ in T_{4q+1}). Using $A(3,5) = 5$ (see Lemma 3.2), we get $A(3,4q+1) \leq qA(3,5) + 4q(q-1) = 4q^2 + q = \frac{(4q+1)4q}{4}$ for $q \equiv 0$ or $1 \pmod 3$. So $A(3,n) \leq \frac{n(n-1)}{4}$ for $n \equiv 1$ or $5 \pmod{12}$. □

We group the non-optimal constructions in Proposition 3.8 and Proposition 3.9 according to whether they differ from the lower bound by either a constant or a linear additive term, respectively.

Proposition 3.8
If $n \equiv 8 \pmod{12}$, $A(3,n) \leq \frac{n^2}{4} + 2$.
If $n \equiv 9 \pmod{12}$, $A(3,n) = \frac{n(n-1)}{4} + 3$.

Proof: We start from G of type $2^{q-1}4$ with $q \equiv 1 \pmod 3$, which can be decomposed by Lemma 3.1 into $\frac{2(q-1)(q+2)}{3}$ K_3's. As in the proof of Proposition 3.7, we get a decomposition of T_{4q+4} into $q-1$ T_4's, one T_8 and $\frac{2(q-1)(q+2)}{3}$ $\vec{K}_{2,2,2}$'s (indeed, the independent set V_q of G has 4 vertices, so in H it induces an independent set of 8 vertices). So using $A(3,4) = 4$ and $A(3,8) \leq 18$ (see Lemma 3.2), we get $A(3,4q+4) \leq (q-1)A(3,4) + A(3,8) + 4(q-1)(q+2) \leq 4q^2 + 8q + 6 = \frac{(4q+4)^2}{4} + 2$ for $q \equiv 1 \pmod 3$, so $A(3,n) \leq \frac{n^2}{4} + 2$ for $n \equiv 8 \pmod{12}$.

Similarly, adding a vertex ∞ to H we get a decomposition of T_{4q+1} into $q-1$ T_5's, one T_9 and $h = \frac{2(q-1)(q+2)}{3}$ K_3's. So using $A(3,5) = 5$ and $A(3,9) \leq 21$ we get $A(3,4q+5) \leq$

$(q-1)A(3,5)+A(3,9)+4(q-1)(q+2) \leq 4q^2+9q+8 = \frac{(4q+5)(4q+4)}{4}+3$ for $q \equiv 1 \pmod 3$, so $A(3,n) \leq \frac{n(n-1)}{4}+3$ for $n \equiv 9 \pmod{12}$. □

Proposition 3.9
If $n \equiv 2 \pmod{12}$, $A(3,n) \leq \frac{n^2}{4}+\frac{n+4}{6}$.
If $n \equiv 3 \pmod{12}$, $A(3,n) \leq \frac{n^2+3}{4}$.
If $n \equiv 6 \pmod{12}$, $A(3,n) \leq \frac{n^2}{4}+\frac{n}{6}$.
If $n \equiv 7 \pmod{12}$, $A(3,n) \leq \frac{n^2-1}{4}$.
If $n \equiv 10 \pmod{12}$, $A(3,n) \leq \frac{n^2}{4}+\frac{n+8}{6}$.
If $n \equiv 11 \pmod{12}$, $A(3,n) \leq \frac{n^2+3}{4}+\varepsilon$, *with* $\varepsilon = 1$ *for* $n = 11, 35$.

Proof: We use as graph G of Lemma 3.1 a multipartite graph of type $3^{q-1}u$ with $3(q-1)+u$ vertices, in order to get a decomposition of $T_{6(q-1)+2u}$ (resp. $T_{6(q-1)+2u+1}$) into $q-1$ T_6's (resp. T_7's), one T_{2u} (resp. T_{2u+1}) and the digraph H itself decomposed by Lemma 3.1 into $h = \frac{9(q-1)(q-2)}{6}+u(q-1)$ $\vec{K}_{2,2,2}$'s. We distinguish several cases according to the value of u.

Case 1: $u = 1$, $q \geq 3$ odd.

Let $n \equiv 2 \pmod{12}$, $n = 6q-4$. Using that $A(3,2) = 2$ and $A(3,6) \leq 10$ we get $A(3,6q-4) \leq (q-1)A(3,6)+A(3,2)+(q-1)(9q-12) \leq 9q^2-11q+4 = \frac{(6q-4)^2}{4}+q = \frac{n^2}{4}+\frac{n+4}{6}$.

Let $n \equiv 3 \pmod{12}$, $n = 6q-3$. Using that $A(3,3) = 3$ and $A(3,7) \leq 12$ we get $A(3,6q-3) \leq (q-1)A(3,7)+A(3,3)+(q-1)(9q-12) \leq 9q^2-9q+3 = \frac{(6q-3)^2}{4}+\frac{3}{4} = \frac{n^2+3}{4}$.

Case 2: $u = 3$, $q \geq 3$ odd.

Let $n \equiv 6 \pmod{12}$, $n = 6q$. Using that $A(3,6) \leq 10$ we get $A(3,6q) \leq qA(3,6)+9q(q-1) \leq 9q^2+q = \frac{n^2}{4}+\frac{n}{6}$.

Let $n \equiv 7 \pmod{12}$, $n = 6q+1$. Using that $A(3,7) \leq 12$ we get $A(3,6q+1) \leq qA(3,7)+9q(q-1) \leq 9q^2+3q = \frac{n^2-1}{4}$.

Case 3: $u = 5$, $q \geq 5$ odd.

Let $n \equiv 10 \pmod{12}$, $n = 6q+4$. Using that $A(3,6) \leq 10$ and $A(3,10) \leq 28$ we get $A(3,6q+4) \leq (q-1)A(3,6)+A(3,10)+(q-1)(9q+12) \leq 9q^2+13q+6 = \frac{(6q+4)^2}{4}+\frac{6q+12}{6} = \frac{n^2}{4}+\frac{n+8}{6}$.

Let $n \equiv 11 \pmod{12}$, $n = 6q+5$. Using that $A(3,7) \leq 12$ and $A(3,11) \leq 31$ we get $A(3,6q+5) \leq (q-1)A(3,7)+A(3,11)+(q-1)(9q+12) \leq 9q^2+15q+7 = \frac{n^2+3}{4}$.

For $q = 23$ we have $A(3,23) \leq 132 = \frac{23^2-1}{4}$, one less than the value given by the preceding construction. Using $u = 11$, $q \geq 7$ odd, $n = 6q+17$, $A(3,7) \leq 12$, and $A(3,23) \leq 132$ we get $A(3,6q+17) \leq (q-1)A(3,7)+A(3,23)+(q-1)(9q+48) \leq 9q^2+51q+72 = \frac{(6q+17)^2-1}{4} = \frac{n^2-1}{4}$. It might be that $A(3,11) \leq 30$, and then the bound $\frac{n^2-1}{4}$ would be also attained for $n = 11$ and 35. □

3.6 Case $C > 3$

For $C > 3$, we distinguish two cases according to whether C is of the form $\frac{k(k+1)}{2}$ or not. We focus on those cases in Sections 3.6.1 and 3.6.2.

3.6.1 C not of the form $k(k+1)/2$

If C is not of the form $\frac{k(k+1)}{2}$, we can improve the lower bound of Theorem 3.1, as we did for $C = 2$ in Proposition 3.4. We provide the details for $C = 4$ and sketch the ideas for $C = 5$, that show how to improve the lower bound for any value of C not of the form $k(k+1)/2$.

Proposition 3.10

$$A(4,n) \;\geq\; \frac{7}{32}n(n-1) \;=\; \left(\frac{3}{14}+\frac{1}{224}\right)n(n-1).$$

Proof: The values of $\gamma(4,p)$ are given in Table 3.1, so Equation (3.11) becomes in the case $C = 4$

$$A = \sum_{p=2}^{n} pa_p \;\geq\; \frac{3}{7}\sum_{p=2}^{n} a_p\gamma(4,p)+\frac{11}{7}a_2+\frac{12}{7}a_3+\frac{10}{7}a_4+\frac{5}{7}a_5+\frac{3}{7}a_6+\frac{1}{7}(a_7+2a_8+a_{10}+2a_{11}+a_{13}+2a_{14}+\ldots). \tag{3.25}$$

Using that $\sum_{p=2}^{n} a_p\gamma(4,p) \geq \frac{n(n-1)}{2}$, Equation (3.25) becomes

$$14A \;\geq\; 3n(n-1)+22a_2+24a_3+20a_4+10a_5+6a_6+2a_7+4a_8+\ldots \tag{3.26}$$

On the other hand,

$$A \;\geq\; 9\left(\Lambda-\sum_{i=2}^{8} a_i\right)+\sum_{i=2}^{8} i\cdot a_i \;=\; 9\Lambda-7a_2-6a_3-5a_4-4a_5-3a_6-2a_7-a_8. \tag{3.27}$$

Summing Equations (3.26) and (3.27) and using that $\Lambda \geq \frac{n(n-1)}{32}+\frac{n-1}{32}$ by Proposition 3.1 yields

$$15A \;\geq\; \frac{105}{32}n(n-1)+\frac{9}{32}(n-1), \quad \text{and therefore} \quad A \;\geq\; \frac{7}{32}n(n-1)+\frac{3}{160}(n-1).$$

□

For $C = 5$, a similar computation with $\rho(5) = 8/3$ gives

$$8A \;\geq\; \frac{3}{2}n(n-1)+13a_2+15a_3+14a_4+10a_5+3a_6+2a_7+a_8. \tag{3.28}$$

$$A \;\geq\; 9\Lambda-7a_2-6a_3-5a_4-4a_5-3a_6-2a_7-a_8. \tag{3.29}$$

So again, Summing Equations (3.28) and (3.29) and using that $\Lambda \geq \frac{n(n-1)}{40}+\frac{n-1}{40}$ by Proposition 3.1 yields

$$A \;\geq\; \frac{n(n-1)}{6}+\frac{n(n-1)}{40}+\frac{n-1}{40} \;=\; \frac{23}{120}n(n-1)+\frac{n-1}{40} \;=\; \left(\frac{3}{16}+\frac{1}{240}\right)n(n-1)+\frac{n-1}{40}.$$

3.6.2 C of the form $k(k+1)/2$

For $C = \frac{k(k+1)}{2}$ the lower bound of Theorem 3.1 can be attained, according to the existence of a type of *k-GDD*, called *Balanced Incomplete Block Design* (*BIBD*). A $(v, k, 1)$-*BIBD* consists simply of a partition of K_v into K_k's.

Theorem 3.5 *If there exists a $(k+1)$-GDD of type k^q (that is, a decomposition of $K_{k\times q}$ into K_{k+1}'s), then there exists an optimal admissible partition of T_{2kq+1} for $C = \frac{k(k+1)}{2}$ with $\frac{n(n-1)}{2k}$ ADMs.*

Proof: The lower bounds follows from Theorem 3.1. For the upper bound, as we did in Proposition 3.7 (case $k = 2$, $C = 2$), we replace each vertex i of $K_{k\times q}$ with two vertices i_A and i_B, and add a new vertex ∞. We label the vertices of the obtained T_{2kq+1} with $\infty, 1_A, \ldots, (kq)_A, 1_B, \ldots, (kq)_B$. To each K_{k+1} of the decomposition of $K_{k\times q}$ we associate a $T_{2\times(k+1)}$, which is an optimal digraph for $C = \frac{k(k+1)}{2}$ with $2(k+1)$ vertices and $2k(k+1)$ edges, hence attaining $\rho(C) = k$. So adding vertex ∞ to the stable sets of size $2k$ we obtain a decomposition of T_{2kq+1} into q T_{2k+1}'s (which are also optimal) and $T_{2\times(k+1)}$'s.

If $K_{k\times q}$ is decomposable into K_{k+1}'s, the number of K_{k+1}'s (and so the number of $T_{2\times(k+1)}$'s) is $\frac{kq(q-1)}{k+1}$. Therefore the total number of ADMs is $q(2k+1) + 2kq(q-1) = \frac{(2kq+1)2kq}{2k} = \frac{n(n-1)}{2k}$. □

Note that a decomposition of $K_{k\times q}$ into K_{k+1}'s is equivalent to a decomposition of K_{kq+1} into K_{k+1}'s by adding a new vertex ∞, that is, a $(kq+1, k+1, 1)$-*BIBD*. In particular, such designs are known to exist if n is large enough and $(kq+1)kq \equiv 0 \pmod{k(k+1)}$ [75]. For example, for $k = 3$ and $q \equiv 0$ or $1 \pmod 4$, or $k = 4$ and $q \equiv 0$ or $1 \pmod 5$.

Corollary 3.1
If $C = 6$ and $n \equiv 1$ or $7 \pmod{24}$, $A(6, n) = \frac{n(n-1)}{6}$.
If $C = 10$ and $n \equiv 1$ or $9 \pmod{40}$, $A(10, n) = \frac{n(n-1)}{8}$.

Corollary 3.2 *For $C \in \{15, 21, 28, 36\}$, there exists a small set of values of n for which the existence of a BIBD remains undecided (179 values overall, see [75, pages 73-74]). For the values of n different from these undecided BIBDs, the following results apply.*
If $C = 15$ and $n \equiv 1$ or $11 \pmod{30}$, $A(15, n) = \frac{n(n-1)}{10}$.
If $C = 21$ and $n \equiv 1$ or $13 \pmod{84}$, $A(21, n) = \frac{n(n-1)}{12}$.
If $C = 28$ and $n \equiv 1$ or $15 \pmod{112}$, $A(28, n) = \frac{n(n-1)}{14}$.
If $C = 36$ and $n \equiv 1$ or $17 \pmod{144}$, $A(36, n) = \frac{n(n-1)}{16}$.

Wilson proved [194] that for v large enough, K_v can be decomposed into subgraphs isomorphic to any given graph G, if the trivial necessary conditions about the degree and the number of edges are satisfied. Thus, we can assure that optimal constructions exist when $C = \frac{k(k+1)}{2}$ for all $k > 0$.

Corollary 3.3 *If $C = \frac{k(k+1)}{2}$, then $A(C, n) = \frac{n(n-1)}{2k}$ for $n \equiv 1$ or $2k+1 \pmod{4C}$ large enough.*

We can also use decompositions of $K_{p\times q}$ into K_{k+1}'s to get constructions asymptotically optimal, but not attaining the lower bound like for $C = 3$. For instance, for $C = 6$ the proof of Theorem 3.5 gives (without adding the vertex ∞) that for $q \equiv 0$ or $1 \pmod 4$ and $n \equiv 0$ or $6 \pmod{24}$,

$$A(6, 6q) \;\leq\; qA(6, 6) + 6q(q-1) \;=\; 6q^2 \;=\; \frac{n^2}{6}.$$

That might be an optimal value if we could improve the lower bound for $C = 6$ as we did for $C = 3$ in Proposition 3.6, but the calculations become considerably more complicated.

Corollary 3.4
For $n \equiv 0$ *or* $6 \pmod{24}$, $\frac{n(n-1)}{6} \;\leq\; A(6, n) \;\leq\; \frac{n^2}{6}$.
For $n \equiv 0$ *or* $8 \pmod{40}$, $\frac{n(n-1)}{8} \;\leq\; A(10, n) \;\leq\; \frac{n^2}{8}$.

For a general C of the form $C = \frac{k(k+1)}{2}$, the improved lower bound one could expect is $\frac{n^2}{2k}$.

3.7 Unidirectional or Bidirectional Rings?

This section is devoted to compare unidirectional and bidirectional rings in terms of minimizing electronics cost, when these rings are used in a WDM network with traffic grooming. In [47] general lower bounds are given in unidirectional rings:

$$A_{uni}(C, n) \geq \frac{n(n-1)}{2}\frac{1}{\eta(C)},$$

$$\text{where } \eta(C) = \begin{cases} \frac{k-1}{2}, & \text{if } \frac{(k-1)k}{2} \leq C \leq \frac{(k-1)(k+1)}{2} \\ \frac{C}{k+1}, & \text{if } \frac{(k-1)(k+1)}{2} \leq C \leq \frac{k(k+1)}{2} \end{cases}$$

In this chapter we have provided a general lower bound for all values of C and n in bidirectional rings (taking into account requests both clockwise and counterclockwise):

$$A_{bi}(C, n) \geq \frac{n(n-1)}{2}\frac{2k}{(k-1)k + r},$$

where $C = 1 + 2 + \ldots + k - 1 + r = \frac{(k-1)k}{2} + r$, with $0 \leq r < k$.

The first observation is that both bounds behave like $\binom{n}{2}$ on n, and like $\frac{1}{\sqrt{C}}$ on C. Furthermore, the bounds coincide when $C = (k-1)k/2$ for all $k > 0$. It is straightforward to prove that for all other values of C the unidirectional ring bound is strictly larger. This is what one can expect a priori, because the set of possible routings in a bidirectional ring contains unidirectional routing as a particular case. In this chapter we have focused on symmetric routing following a shortest path in the bidirectional ring, so this result gives indeed important information. We depict both bounds in the first graph of Figure 3.7. In this figure the bounds are normalized by $n(n-1)/2$ and the horizontal axis is logarithmic

on C. The solid (resp. dashed) line represents the bidirectional (resp. unidirectional) bound.

A natural question is the following: does it exist a construction for bidirectional rings with cost strictly smaller that the lower bound for unidirectional rings? Indeed, consider $C = 2$, and then the ratio between both bounds is $12/11 = 1,0909$. That is, if we have an α-approximation for bidirectional rings with $\alpha < 12/11$, the cost of this solution is strictly smaller than the lower bound for unidirectional rings. The 34/33-approximation provided in Section 3.4.2 is such an example. Thus, we conclude that the cost in bidirectional rings with shortest path routing, in terms of number of ADMs needed, can be strictly smaller than the cost in unidirectional rings.

Because of Wilson's theorem [194], both lower bounds are achieved for infinite values of C and n. Thus, it makes sense to compare these bounds to infer which type of routing is better in terms of electronics cost. Let us consider the ratio between both bounds, namely $A^*_{uni}(C,n)$ and $A^*_{bi}(C,n)$. If $\frac{(k-1)k}{2} \leq C \leq \frac{(k-1)(k+1)}{2}$:

$$1 \leq \frac{A^*_{uni}(C,n)}{A^*_{bi}(C,n)} = \frac{(k-1)k+r}{(k-1)k} = 1 + \frac{r}{(k-1)k} \leq 1 + \frac{1}{k}$$

Finally, if $\frac{(k-1)(k+1)}{2} \leq C \leq \frac{k(k+1)}{2}$:

$$1 \leq \frac{A^*_{uni}(C,n)}{A^*_{bi}(C,n)} = \frac{k+1}{C}\frac{(k-1)k+r}{2k} = \left(1+\frac{1}{k}\right)\left(1-\frac{r}{2C}\right) \leq 1 + \frac{1}{k}$$

That is, in all cases we have that

$$1 \leq \frac{A^*_{uni}(C,n)}{A^*_{bi}(C,n)} \leq 1 + \frac{1}{k} \tag{3.30}$$

From Equation (3.30) we conclude that

$$\lim_{C\to\infty} \frac{A^*_{uni}(C,n)}{A^*_{bi}(C,n)} = 1, \qquad \forall\ n > 0,$$

as we can see in the second graph of Figure 3.7.

Hence, for big values of n there is no real improvement in bidirectional rings, in terms of cost of electronic switching. Is this conclusion surprising? In fact, our objective function it the number of electronic terminations, that is, the number of ADMs. And, roughly speaking, we need to place an ADM in a node when a request either originates or terminates, regardless of routing. Thus, it is not that surprising that, asymptotically, bidirectional routing does not improve unidirectional routing. Of course, there is a drawback when using unidirectional rings. This drawback is the bandwidth utilization. Let us say that a request routed along a path of length l uses l *units of bandwidth*. In the all-to-all case, in a unidirectional ring on n nodes each pair of communicating nodes uses n units of bandwidth, since the requests (i,j) and (j,i) are routed via disjoint paths. Thus, the total bandwidth utilization is $\binom{n}{2} \cdot n$, and the load of each link is $\frac{n(n-1)}{2}$. On the other hand, in a bidirectional ring it is easy to compute the load of a link, that turns out to be $\frac{n^2-1}{4}$ for n odd. That is, the bandwidth utilization is asymptotically twice better in bidirectional rings. In conclusion, there is a trade-off between bandwidth utilization (better in bidirectional rings) and technological simplicity (better in unidirectional rings). But, concerning the electronics cost, both rings behave in a similar fashion.

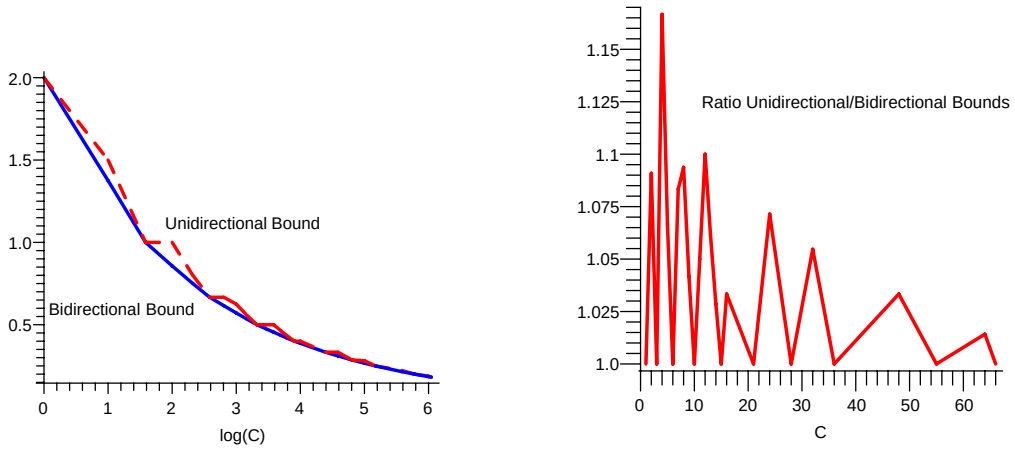

Figure 3.7: Comparison of lower bounds for unidirectional and bidirectional rings.

3.8 Conclusions

In this chapter we studied the minimization of ADMs in optical WDM bidirectional ring networks under the assumption of symmetric shortest path routing and all-to-all unitary requests. We precisely formulated the problem in terms of graph decompositions, and stated a general lower bound for all the values of C and n. We then studied extensively the cases $C = 2$ and $C = 3$, providing improved lower bounds, optimal constructions for several infinite families, as well as asymptotically optimal constructions and approximations. To the best of our knowledge, these are the first optimal solutions in the literature for traffic grooming in bidirectional rings. We then study the case $C > 3$, focusing specifically on the case $C = k(k+1)/2$ for some $k \geq 1$. We gave optimal decompositions for several congruence classes of n, using the existence of a certain combinatorial design. We concluded with a comparison of the switching cost in unidirectional and bidirectional WDM rings.

Further research is needed to find new families of optimal solutions for other values of C. The first step should be to improve the general lower bound for other values of C, namely, finding a closed formula. It would be interesting to consider other kinds of routing in bidirectional rings, not necessarily symmetric or using shortest paths. Stating which kind of routing is the best for each value of n and C would be a nice result. Finally, studying the traffic grooming problem using graph partitioning tools in other topologies, like trees or hypercubes, would be also interesting.

Chapter 4

Two-period Grooming

In this chapter we study grooming for two-period optical networks, a variation of the traffic grooming problem for WDM ring networks introduced by Colbourn, Quattrocchi, and Syrotiuk [78, 79]. In the two-period grooming problem, during the first period of time, there is all-to-all uniform traffic among n nodes, each request using $1/C$ of the bandwidth; and during the second period, there is all-to-all uniform traffic only among a subset V of v nodes, each request now being allowed to use $1/C'$ of the bandwidth, where $C' < C$. We determine the minimum drop cost (minimum number of ADMs) for any n, v and $C = 4$ and $C' \in \{1, 2, 3\}$. To do this, we use tools of graph decompositions. Indeed the two-period grooming problem corresponds to minimizing the total number of vertices in a partition of the edges of the complete graph K_n into subgraphs, where each subgraph has at most C edges and where furthermore it contains at most C' edges of the complete graph on v specified vertices. Subject to the condition that the two-period grooming has the least drop cost, the minimum number of wavelengths required is also determined in each case.

Keywords: traffic grooming, SONET ADM, optical networks, graph decomposition, design theory.

4.1 Introduction

In this chapter we deal with the traffic grooming problem for a unidirectional SONET ring with n nodes, grooming ratio C, and all-to-all uniform unitary traffic. This problem has been modeled as a graph partition problem in both [47] and [136]. In the all-to-all case the set of requests is modeled by the complete graph K_n. To a wavelength λ is associated a subgraph B_λ in which each edge corresponds to a pair of symmetric requests (that is, a circle) and each node to an ADM. The grooming constraint, i.e., the fact that a wavelength can carry at most C requests, corresponds to the fact that the number of edges $|E(B_\lambda)|$ of each subgraph B_λ is at most C. The cost corresponds to the total number of vertices used in the subgraphs, and the objective is therefore to minimize this number.

Traffic Grooming in Unidirectional Rings with All-to-all Traffic
Input: Two integers n and C.
Output: Partition $E(K_n)$ into subgraphs B_λ, $1 \le \lambda \le \Lambda$, s.t. $|E(B_\lambda)| \le C$ for all λ.
Objective: Minimize $\sum_{\lambda=1}^{\Lambda} |V(B_\lambda)|$.

With the all-to-all set of requests, optimal constructions for a given grooming ratio C have been obtained using tools of graph and design theory [75]. See Section II.3.2 (page 30) for a survey of the results in unidirectional rings with all-to-all traffic. Graph decompositions have been extensively studied for other reasons as well. See [61] for an excellent survey, [81] for relevant material on designs with blocksize three, and [75] for terminology in design theory.

Most of the papers on grooming deal with a single (static) traffic matrix. Some articles consider variable (dynamic) traffic, such as finding a solution which works for the maximum traffic demand [55, 199] or for all request graphs with a given maximum degree [C13, C15], but all keep a fixed grooming factor. In [79] an interesting variation of the traffic grooming problem, grooming for two-period optical networks, has been introduced in order to capture some dynamic nature of the traffic. Informally, in the two-period grooming problem each time period supports different traffic requirements. During the first period of time there is all-to-all uniform traffic among n nodes, each request using $1/C$ of the bandwidth; but during the second period there is all-to-all traffic only among a subset V of v nodes, each request now being allowed to use a larger fraction of the bandwidth, namely $1/C'$ where $C' < C$.

Denote by X the subset of n nodes. Therefore the two-period grooming problem can be expressed as follows.

Two-Period Grooming in the Ring
Input: Four integers n, v, C, and C'.
Output: A partition (denoted $N(n, v; C, C')$) of $E(K_n)$ into subgraphs B_k, $1 \le \lambda \le \Lambda$, such that for all λ, $|E(B_\lambda)| \le C$, and $|E(B_\lambda) \cap (V \times V)| \le C'$, with $V \subseteq X$, $|V| = v$.
Objective: Minimize $\sum_{\lambda=1}^{\Lambda} |V(B_\lambda)|$.

Following [78], a grooming is denoted by $N(n, C)$. When the grooming $N(n, C)$ is *optimal*, i.e., minimizes the total ADM cost, then the grooming is denoted by $\mathscr{ON}(n, C)$. Whether general or optimal, the drop cost of a grooming is denoted by *cost* $N(n, C)$ or *cost* $\mathscr{ON}(n, C)$, respectively.

A grooming of a two-period network $N(n, v; C, C')$ with grooming ratios (C, C') coincides with a graph decomposition $(X, \mathcal{B})$ of K_n (using standard design theory terminology, $\mathcal{B}$ is the set of all the *blocks* of the decomposition) such that $(X, \mathcal{B})$ is a grooming $N(n, C)$ in the first time period, and $(X, \mathcal{B})$ faithfully embeds a graph decomposition of K_v such that $(V, \mathcal{D})$ is a grooming $N(v, C')$ in the second time period. Let $V \subseteq X$. The graph decomposition $(X, \mathcal{B})$ *embeds* the graph decomposition $(V, \mathcal{D})$ if there is a mapping $f : \mathcal{D} \to \mathcal{B}$ such that D is a subgraph of $f(D)$ for every $D \in \mathcal{D}$. If f is injective (i.e., one-to-one), then $(X, \mathcal{B})$ *faithfully embeds* $(V, \mathcal{D})$. This concept of faithfully embedding has been explored in [77, 170].

We use the notation $\mathscr{ON}(n, v; C, C')$ to denote an optimal grooming $N(n, v; C, C')$.

As it turns out, an $\mathscr{ON}(n, v; C, C')$ does not always coincide with an $\mathscr{ON}(n, C)$. Generally we have *cost* $\mathscr{ON}(n, v; C, C') \geq$ *cost* $\mathscr{ON}(n, C)$ (see Examples 4.2 and 4.3). Of particular interest is the case when *cost* $\mathscr{ON}(n, v; C, C') =$ *cost* $\mathscr{ON}(n, C)$ (see Example 4.1).

Example 4.1 *Let $n = 7$, $v = 4$, $C = 4$. Let $V = \{0, 1, 2, 3\}$ and $W = \{a_0, a_1, a_2\}$. An optimal decomposition is given by the three triangles $(a_0, 0, 1)$, $(a_1, 1, 2)$, and $(a_2, 2, 3)$, and the three 4-cycles $(0, 2, a_0, a_1)$, $(0, 3, a_0, a_2)$, and $(1, 3, a_1, a_2)$, giving a total cost of 21 ADMs.*

This solution is valid and optimal for both $C' = 1$ and $C' = 2$, and it is optimal for the classical Traffic Grooming in the Ring *problem when $n = 7$ and $C = 4$. Therefore, cost $\mathscr{ON}(7, 4; 4, 1) =$ cost $\mathscr{ON}(7, 4; 4, 2) =$ cost $\mathscr{ON}(7, 4) = 21$.*

Example 4.2 *Let $n = 7$, $v = 5$, $C = 4$, and $C' = 2$. Let $V = \{0, 1, 2, 3, 4\}$ and $W = \{a_0, a_1\}$. We see later that an optimal decomposition is given by the five kites $(a_0, 1, 2; 0)$, $(a_0, 3, 4; 1)$, $(a_1, 1, 3; 2)$, $(a_1, 2, 4; 0)$ and $(a_0, a_1, 0; 1)$, plus the edge $\{0, 3\}$, giving a total cost of 22 ADMs. So cost $\mathscr{ON}(7, 5; 4, 2) = 22$. Note that this decomposition is not a valid solution for $C' = 1$, since there are subgraphs containing more than one edge with both end-vertices in V.*

Example 4.3 *Let $n = 7$, $v = 5$, $C = 4$, and $C' = 1$. Let again $V = \{0, 1, 2, 3, 4\}$ and $W = \{a_0, a_1\}$. We see later that an optimal decomposition is given by the four K_3s $(a_0, 1, 2)$, $(a_0, 3, 4)$, $(a_1, 0, 3)$, and $(a_1, 2, 4)$, the C_4 $(0, 1, a_1, a_0)$, plus the five edges $\{0, 4\}$, $\{1, 3\}$, $\{0, 2\}$, $\{1, 4\}$, and $\{2, 3\}$, giving a total cost of 26 ADMs. So cost $\mathscr{ON}(7, 5; 4, 1) = 26$.*

Colbourn, Quattrocchi, and Syrotiuk [78, 79] completely solved the cases when $C = 2$ and $C = 3$ ($C' = 1$ or 2). In this chapter we determine the minimum drop cost of an $N(n, v; 4, C')$ for all $n \geq v \geq 0$ and $C' \in \{1, 2, 3\}$.

We are also interested in determining the minimum number of wavelengths, or *wavecost*, required in an assignment of wavelengths to a decomposition. Among the $\mathscr{ON}(n, 4)$s one having the minimum wavecost is denoted by $\mathscr{MON}(n, 4)$, and the corresponding minimum number of wavelengths by *wavecost*$\mathscr{MON}(n, 4)$. We characterize the $\mathscr{ON}(n, v; C, C')$ whose wavecost is minimum among all $\mathscr{ON}(n, v; C, C')$s, and which is denoted by $\mathscr{MON}(n, v; C, C')$; the wavecost is itself denoted by *wavecost*$\mathscr{MON}(n, v; C, C')$.

We deal separately with each value of $C' \in \{1, 2, 3\}$. Table 4.1 summarizes the cost formulas for $n = v + w > 4$.

4.2 Preliminaries

We establish some notation to be used throughout the chapter. K_n denotes a complete graph on n vertices and K_X represents the complete graph on the vertex set X. A triangle with edges $\{\{x, y\}, \{x, z\}, \{y, z\}\}$ is denoted by (x, y, z). A 4-cycle with edges $\{\{x, y\}, \{y, z\}, \{z, u\}, \{u, x\}\}$ is denoted by (x, y, z, u). A *kite* with edges $\{\{x, y\}, \{x, z\}, \{y, z\}, \{z, u\}\}$ is denoted by $(x, y, z; u)$. The groomings to be produced also employ paths; the path on k

$$cost\ \mathscr{ON}(v+w, v; 4, 1) = \begin{cases} \binom{v+w}{2} & \text{if } v \le w+1 \\ \binom{v+w}{2} + \binom{v}{2} - \left\lfloor \frac{vw}{2} \right\rfloor & \text{if } v \ge w+1 \end{cases}$$

$$cost\ \mathscr{ON}(v+w, v; 4, 2) = \begin{cases} \binom{v+w}{2} & \text{if } v \le 2w \\ \binom{v+w}{2} + \left\lceil \frac{1}{2}\binom{v}{2} \right\rceil - \frac{vw}{2} + \delta & \text{if } v > 2w \text{ and } v \text{ even} \\ \quad \text{where } \delta = \begin{cases} 1 & \text{if } w = 2, \text{ or} \\ & \text{if } w = 4 \text{ and} \\ & \quad v \equiv 0 \pmod 4 \\ 0 & \text{otherwise} \end{cases} & \\ \binom{v+w}{2} + \left\lceil \frac{1}{2}\left(\binom{v}{2} - vw - \left\lceil \frac{w}{2} \right\rceil\right) \right\rceil + \delta & \text{if } v > 2w \text{ and } v \text{ odd} \\ \quad \text{where } \delta = \begin{cases} 1 & \text{if } w = 3 \text{ and} \\ & \quad v \equiv 3 \pmod 4 \\ 0 & \text{otherwise} \end{cases} & \end{cases}$$

$$cost\ \mathscr{ON}(v+w, v; 4, 3) = \binom{v+w}{2}$$

Table 4.1: Cost formulas for $n = v + w > 4$.

vertices P_k is denoted by $[x_1, \ldots, x_k]$ when it contains edges $\{x_i, x_{i+1}\}$ for $1 \le i < k$. Now let $G = (X, E)$ be a graph. If $|X|$ is even, a set of $|X|/2$ disjoint edges in E is a *1-factor*; a partition of E into 1-factors is a *1-factorization*. Similarly, if $|X|$ is odd, a set of $(|X|-1)/2$ disjoint edges in E is a *near 1-factor*; a partition of E into near 1-factors is a *near 1-factorization*. We also employ well-known results on partial triple systems and group divisible designs with block size three; see [81] for background.

The vertices of the set V are the integers modulo v denoted by $0, 1, \ldots, v-1$. The vertices not in V, that is in $X \setminus V$, forms the set W of size $w = n - v$ and is denoted by $a_0, \ldots, a_{w-1}$, the indices being taken modulo w.

Among graphs with three or fewer edges (i.e., when $C = 3$),the only graph with the minimum ratio (number of vertices over the number of edges) is the triangle. For $C = 4$ three different such graphs have minimum ratio 1: the triangle, the 4-cycle, and the kite. This simplifies the problem substantially. Indeed, in contrast to the lower bounds in [79], in this case the lower bounds arise from easy classification of the edges on V. We recall the complete characterization for optimal groomings with a grooming ratio of four:

Theorem 4.1 [47, 145] *cost* $\mathscr{ONS}(4,4) = 7$ *and, for* $n \ge 5$, *cost* $\mathscr{ONS}(n,4) = \binom{n}{2}$. *Furthermore a* $\mathscr{MONS}(4,4)$ *employs two wavelengths and can be realized by a kite and a* P_3 *(or a* K_3 *and a star), and a* $\mathscr{MONS}(n,4)$, $n \ge 5$, *employs* $\left\lceil \frac{n(n-1)}{8} \right\rceil$ *wavelengths and can be realized by* t K_3*s and* $\left\lceil \frac{n(n-1)}{8} - t \right\rceil$ *4-cycles or kites, where*

$$t = \begin{cases} 0 & \text{if } n \equiv 0,1 \pmod 8 \\ 1 & \text{if } n \equiv 3,6 \pmod 8 \\ 2 & \text{if } n \equiv 4,5 \pmod 8 \\ 3 & \text{if } n \equiv 2,7 \pmod 8 \end{cases}.$$

In order to unify the treatment of the lower bounds, in a decomposition $N(v+w, v; 4, C')$ for $C' \in \{1, 2\}$, we call an edge with both ends in V *neutral* if it appears in a triangle, 4-cycle, or kite; we call it *positive* otherwise. An edge with one end in V and one in W is a *cross edge*.

Lemma 4.1

1. *In an* $N(v+w, v; 4, C')$ *with* $C' \in \{1, 2\}$, *the number of neutral edges is at most* $\frac{1}{2}C'vw$.
2. *When* v *is odd and* $C' = 2$, *the number of neutral edges is at most* $vw - \frac{w}{2}$.

Proof: Every neutral edge appears in a subgraph having at least two cross edges. Thus the number of subgraphs containing one or more neutral edges is at most $\frac{1}{2}vw$. Each can contain at most C' neutral edges, and hence there are at most $\frac{1}{2}C'vw$ neutral edges. This proves the first statement.

Suppose now that $C' = 2$ and v is odd. Any subgraph containing two neutral edges employs exactly two cross edges incident to the same vertex in W. Thus the number α of such subgraphs is at most $\frac{1}{2}w(v-1)$. Then remaining neutral edges must arise (if present) in triangles, kites, or 4-cycles that again contain two cross edges but only one neutral edge; their number, β, must satisfy $\beta \le \frac{vw}{2} - \alpha$. Therefore the number of neutral edges, $2\alpha + \beta$, satisfies $2\alpha + \beta \le \frac{1}{2}w(v-1) + \frac{vw}{2} = vw - \frac{w}{2}$. □

When $C = 3$ there are strong interactions among the decompositions placed on V, on W, and on the cross edges [78,79]; fortunately here we shall see that the structure on V suffices to determine the lower bounds. Because every $N(v+w,v;4,C')$ is an $N(v+w,v;4,C'+1)$ for $1 \le C' \le 3$, and $N(v+w,v;4,4)$ coincides with $N(v+w,4)$, $cost\ \mathscr{ON}(v+w,v;4,1) \ge cost\ \mathscr{ON}(v+w,v;4,2) \ge cost\ \mathscr{ON}(v+w,v;4,3) \ge cost\ \mathscr{ON}(v+w,4)$. We use these obvious facts to establish lower and upper bounds without further comment.

4.3 Case $C' = 1$

4.3.1 $\mathscr{ON}(n,v;4,1)$

Theorem 4.2 *Let $n = v + w \ge 5$.*

1. *$cost\ \mathscr{ON}(v+w,v;4,1) = cost\ \mathscr{ON}(v+w,4)$ when $v \le w+1$.*
2. *$cost\ \mathscr{ON}(v+w,v;4,1) = \binom{v+w}{2} + \binom{v}{2} - \lfloor \frac{vw}{2} \rfloor$ when $v \ge w+1$.*

Proof: To prove the lower bound, we establish that $cost\ \mathscr{ON}(v+w,v;4,1) \ge \binom{v+w}{2} + \binom{v}{2} - \lfloor \frac{vw}{2} \rfloor$. It suffices to prove that the number of subgraphs employed in an $N(v+w,v;4,1)$ other than triangles, kites, and 4-cycles is at least $\lceil \binom{v}{2} - \frac{1}{2}vw \rceil = \binom{v}{2} - \lfloor \frac{1}{2}vw \rfloor$. By Lemma 4.1, this is a lower bound on the number of positive edges in any such decomposition; because each positive edge lies in a different subgraph of the decomposition, the lower bound follows.

Now we turn to the upper bounds. For the first statement, because an $\mathscr{ON}(v+w,v;4,1)$ is also an $\mathscr{ON}(v+w,v-1;4,1)$, it suffices to consider $v \in \{w, w+1\}$. When $v = w$, write $v = 4s+t$ with $t \in \{0,3,5,6\}$. Form on V a complete multipartite graph with s classes of size four and one class of size t. Replace edge $e = \{x,y\}$ of this graph by the 4-cycle (x,y,a_x,a_y). On $\{x_1,\ldots,x_\ell,a_{x_1},\ldots,a_{x_\ell}\}$ whenever $\{x_1,\ldots,x_\ell\}$ forms a class of the multipartite graph, place a decomposition that is optimal for drop cost and uses 4, 7, 12, and 17 wavelengths when ℓ is 3, 4, 5, or 6, respectively (see Section 4.6.1).

Now let $v = w+1$. Let $V = \{0,\ldots,v-1\}$ and $W = \{a_0,\ldots,a_{v-2}\}$. Form triangles $(i,i+1,a_i)$ for $0 \le i < v-1$. Then form 4-cycles $(i,j+1,a_i,a_j)$ for $0 \le i < j \le v-2$.

Finally, suppose that $v \ge w+2$. When v is even, form a 1-factorization $F_0,\ldots,F_{v-2}$ on V. For $0 \le i < w$, let $\{e_{ij} : 1 \le j \le \frac{v}{2}\}$ be the edges of F_i, and form triangles $T_{ij} = \{a_i\} \cup e_{ij}$. Now for $0 \le i < w$; $1 \le j \le \lfloor \frac{w}{2} \rfloor$; and furthermore $j \ne \frac{w}{2}$ if $i \ge \frac{w}{2}$ and w is even, adjoin edge $\{a_i, a_{i+j \bmod w}\}$ to T_{ij} to form a kite. All edges of 1-factors $\{F_i : w \le i < v-1\}$ are taken as K_2s.

When v is odd, form a near 1-factorization $F_0,\ldots,F_{v-1}$ on V, in which F_{v-1} contains the edges $\{\{2h,2h+1\} : 0 \le h < \frac{v-1}{2}\}$, and near 1-factor F_i misses vertex i for $0 \le i < v$. Then form 4-cycles $(2h,2h+1,a_{2h+1},a_{2h})$ for $0 \le h < \lfloor \frac{w}{2} \rfloor$. For $0 \le i < w$, let $\{e_{ij} : 1 \le j \le \frac{v-1}{2}\}$ be the edges of F_i, and form triangles $T_{ij} = \{a_i\} \cup e_{ij}$. Without loss of generality we assume that $w-1 \in e_{01}$; when w is odd, adjoin $\{w-1, a_{w-1}\}$ to T_{01} to form a kite. Now for $0 \le i < w$; $1 \le j \le \lfloor \frac{w}{2} \rfloor$; and furthermore $j \ne \frac{w}{2}$ if $i \ge \frac{w}{2}$ and w is even and $j \ne 1$ if $i = 2h$ for $0 \le h < \lfloor \frac{w}{2} \rfloor$, adjoin edge $\{a_i, a_{i+j \bmod w}\}$ to T_{ij} to form a kite. All edges of near 1-factors $\{F_i : w \le i < v-1\}$ and the $\frac{v-1}{2} - \lfloor \frac{w}{2} \rfloor$ remaining edges of F_{v-1} are taken as K_2s.

When $v \geq w+1$, each subgraph contains exactly one edge on V and so their number is $\binom{v}{2}$. This fact is later used to prove Theorem 4.4. □

4.3.2 $\mathscr{MON}(n, v; 4, 1)$

Theorem 4.3 *Let $v + w \geq 5$. For $C' = 1$ and $v \leq w$,*

$$wavecost\,\mathscr{MON}(v+w, v; 4, 1) = wavecost\,\mathscr{MON}(v+w, 4).$$

Proof: We need only treat the cases when $v \in \{w, w-1\}$; the case with $v = w$ is handled in the proof of Theorem 4.2. When $v = w - 1$, the argument is identical to that proof, except that we choose $v = 4s + t$ with $t \in \{0, 1, 2, 3\}$ and place decompositions on $\{x_1, \ldots, x_\ell, a_{x_1}, \ldots, a_{x_\ell}, a_v\}$ instead, with 1,3,6,9 wavelengths when $\ell = 1, 2, 3, 4$ respectively (see Section 4.6.2). □

Theorem 4.4 *When $v > w$,*

$$wavecost\,\mathscr{MON}(v+w, v; 4, 1) = \binom{v}{2}.$$

Proof: Since every edge on V appears on a different wavelength, $\binom{v}{2}$ is a lower bound. As noted in the proof of Theorem 4.2 the constructions given there meet this bound. □

The solutions used from Theorem 4.2 are (essentially) the only ones to minimize the number of graphs in an $\mathscr{ON}(v+w, v; 4, 1)$ with $v > w$. However, perhaps surprisingly they are not the only ones to minimize the number of wavelengths. To see this, consider a $\mathscr{ON}(v+w, v; 4, 1)$ with $v > w > 2$ from Theorem 4.2. Remove edges $\{a_0, a_1\}$, $\{a_0, a_2\}$, and $\{a_1, a_2\}$ from their kites, and form a triangle from them. This does not change the drop cost, so the result is also an $\mathscr{ON}(v+w, v; 4, 1)$. It has one more graph than the original. Despite this, it does not need an additional wavelength, since the triangle (a_0, a_1, a_2) can share a wavelength with an edge on V. In this case, while minimizing the number of connected graphs serves to minimize the number of wavelengths, it is not the only way to do so.

4.4 Case $C' = 2$

4.4.1 $\mathscr{ON}(n, v; 4, 2)$

Theorem 4.5 *Let $v + w \geq 5$ and v be even.*

1. *When $v \leq 2w$, cost $\mathscr{ON}(v+w, v; 4, 2) = cost\ \mathscr{ON}(v+w, 4)$.*
2. *When $v \geq 2w+2$, cost $\mathscr{ON}(v+w, v; 4, 2) = \binom{v+w}{2} + \lceil \frac{1}{2}\binom{v}{2} \rceil - \frac{vw}{2} + \delta$, where $\delta = 1$ if $w = 4$ or if $w = 2$ and $v \equiv 0 \pmod 4$, and $\delta = 0$ otherwise.*

Proof: By Lemma 4.1, $\binom{v}{2} - vw$ is a lower bound on the number of positive edges in any $N(v+w, v; 4, 2)$; every subgraph of the decomposition containing a positive edge contains at most two positive edges. So the number of subgraphs employed in an $N(v+w, v; 4, 2)$ other than triangles, kites, and 4-cycles is at least $\lceil \frac{1}{2}\left(\binom{v}{2} - vw\right)\rceil$. The lower bound follows for $w \neq 2, 4$.

As in the proof of Lemma 4.1, denote by α (resp. β) the number of subgraphs containing 2 (resp 1) neutral edges and so at least two cross edges. We have $2\alpha + \beta \leq 2\alpha + 2\beta \leq vw$. Equality in the lower bound, when $v \equiv 0 \pmod 4$, arises only when $\beta = 0$ and therefore to meet the bound an $\mathscr{ON}(w, 4)$ must be placed on W implying that $\delta = 1$ if $w = 2$ or 4. When $v \equiv 2 \pmod 4$, we can have $2\alpha + \beta = vw - 1$ and so $\beta = 1$. We can use an edge on W in a graph with an edge on V. But when $w = 4$, the five edges that would remain on W require drop cost 6, and so $\delta = 1$.

Now we turn to the upper bounds. If $w \geq v - 1$, apply Theorem 4.2. Suppose that $w \leq v - 2$. Let $V = \{0, \ldots, 2t-1\}$ and $W = \{a_0, \ldots, a_{w-1}\}$. Place an $\mathscr{ON}(w, 4)$ on W. Form a 1-factorization on V containing factors $\{F_0, \ldots, F_{w-1}, G_0, \ldots, G_{2t-2-w}\}$ in which the last two 1-factors are $\{\{2h, 2h+1\} : 0 \leq h < t\}$ and $\{\{2h+1, 2h+2 \bmod 2t\} : 0 \leq h < t\}$, whose union is a Hamilton cycle. For $0 \leq i < w$, form triangles T_{ij} by adding a_i to each edge $e_{ij} \in F_i$. For $0 \leq i < \min(w, 2t-1-w)$, observe that $H_i = F_i \cup G_i$ is a 2-factor containing even cycles. Hence there is a bijection σ mapping edges of F_i to edges of G_i so that e and $\sigma(e)$ share a vertex. Adjoin edge $\sigma(e_{ij})$ to the triangle T_{ij} to form a kite. In this way, all edges between V and W appear in triangles or kites, and all edges on V are employed when $v \leq 2w$. When $v \geq 2w + 2$, the edges remaining on V are those of the factors $G_w, \ldots, G_{v-2-w}$.

When $v \neq 2w + 2$, the union of these edges is connected because the union of the last two is connected, and hence it can be partitioned into P_3s (and one P_2 when $v \equiv 2 \pmod 4$) [65, 196]. When $w = 2$ and $v \equiv 2 \pmod 4$, the drop cost can be reduced by 1 as follows. Let $\{x, y\}$ be the P_2 in the decomposition, and let $\{x, z\} \in G_0$. Let T be the triangle obtained by removing $\{x, z\}$ from its kite. Add $\{a_0, a_1\}$ to T to form a kite. Add the P_3 $[y, x, z]$. In this way two isolated P_2s are replaced by a P_3, lowering the drop cost by 1.

When $v = 2w + 2$, we use a variant of this construction. Let R be a graph with vertex set V that is isomorphic to $\frac{v}{4}$ K_4s when $v \equiv 0 \pmod 4$ and to $\frac{v-6}{4}$ K_4s and one $K_{3,3}$ when $v \equiv 2 \pmod 4$. Let $F_1, \ldots, F_{w-1}, G_1, \ldots, G_{w-1}$ be the 1-factors of a 1-factorization of the complement of R (one always exists [179]). Proceed as above to form kites using a_i for $1 \leq i < w$ and the edges of F_i and G_i. For each K_4 of R with vertices $\{p, q, r, s\}$, form kites $(a_0, q, p; r)$ and $(a_0, r, s; p)$. Then add the P_3 $[r, q, s]$. If R contains a $K_{3,3}$ with bipartition $\{\{p, q, r\}, \{s, t, u\}\}$, add kites $(a_0, s, p; t)$, $(a_0, q, t; r)$, and $(a_0, r, u; p)$. What remains is the P_4 $[r, s; q, u]$, which can be partitioned into a P_2 and a P_3. □

In order to treat the odd case, we establish an easy preliminary result:

Lemma 4.2 *Let $w > 3$ be a positive integer. The graph on w vertices containing all edges except for $\lfloor \frac{w}{2} \rfloor$ disjoint edges (i.e., $K_w \setminus \lfloor \frac{w}{2} \rfloor K_2$) can be partitioned into*

1. *4-cycles when w is even;*
2. *kites and 4-cycles when $w \equiv 1 \pmod 4$; and*
3. *kites, 4-cycles, and exactly two triangles when $w \equiv 3 \pmod 4$.*

Proof: Let $W = \{a_0, \ldots, a_{w-1}\}$. When w is even, form 4-cycles $\{(a_{2i}, a_{2j}, a_{2i+1}, a_{2j+1}) : 0 \leq i < j < \frac{w}{2}\}$ leaving uncovered the $\frac{w}{2}$ edges $\{a_{2i}, a_{2i+1}\}$. (This is also a consequence of a much more general result in [127].)

When w is odd, the proof is by induction on w by adding four new vertices. So we provide two base cases for the induction to cover all odd values of w.

For $w = 5$, $K_5 \setminus \{\{a_0, a_1\}, \{a_2, a_3\}\}$ can be partitioned into the two kites $(a_2, a_4, a_0; a_3)$ and $(a_3, a_4, a_1; a_2)$.

For $w = 7$, $K_7 \setminus \{\{a_0, a_1\}, \{a_2, a_3\}, \{a_4, a_5\}\}$ can be partitioned into the kites $(a_3, a_6, a_0; a_5)$, $(a_1, a_6, a_4; a_3)$ and $(a_5, a_6, a_2; a_1)$, and the K_3s (a_0, a_2, a_4) and (a_1, a_3, a_5).

By induction consider an optimal decomposition of $K_w - F$, with $F = \{\{a_{2h}, a_{2h+1}\} : 0 \leq h < \frac{w-1}{2}\}$. Add four vertices $a_w, a_{w+1}, a_{w+2}, a_{w+3}$. Add the C_4s $(a_{2h}, a_w, a_{2h+1}, a_{w+1})$ and $(a_{2h}, a_{w+2}, a_{2h+1}, a_{w+3})$ where $0 \leq h < \frac{w-1}{2}$. Cover the edges of the K_5 on $\{a_{w-1}, a_w, a_{w+1}, a_{w+2}, a_{w+3}\}$ minus the edges $\{a_{w-1}, a_w\}$ and $\{a_{w+1}, a_{w+2}\}$, using two kites as shown for the case when $w = 5$. □

Theorem 4.6 *Let $v + w \geq 5$ and v be odd.*

1. *When $v \leq 2w - 1$, cost $\mathscr{CN}(v + w, v; 4, 2)$ = cost $\mathscr{CN}(v + w, 4)$.*
2. *When $v \geq 2w + 1$, cost $\mathscr{CN}(v + w, v; 4, 2) = \binom{v+w}{2} + \lceil \frac{1}{2} \left(\binom{v}{2} - vw + \lceil \frac{w}{2} \rceil \right) \rceil + \delta$, where $\delta = 1$ if $w = 3$ and $v \equiv 3 \pmod 4$, 0 otherwise.*

Proof: To prove the lower bound, it suffices to prove that the number of subgraphs employed in an $N(v + w, v; 4, 2)$ other than triangles, kites, and 4-cycles is at least $\lceil \frac{1}{2} \left(\binom{v}{2} - vw + \lceil \frac{w}{2} \rceil \right) \rceil$. As in the proof of Theorem 4.5, this follows from Lemma 4.1. When $w = 3$ and $v \equiv 3 \pmod 4$, at least $\binom{v}{2} - 3v + 2$ edges are positive, an even number. To meet the bound, exactly one cross edge remains and exactly two edges on W remain. These necessitate a further graph that is not a triangle, kite, or 4-cycle.

Now we turn to the upper bounds. By Theorem 4.5, cost $\mathscr{CN}((v+1)+(w-1), v+1; 4, 2)$ = cost $\mathscr{CN}(v + w, 4)$ when $v \leq 2w - 3$. So suppose that $v \geq 2w - 1$. Write $v = 2t + 1$.

When $w = t + 1$, form a near 1-factorization on V consisting of $2t + 1$ near 1-factors, $F_0, \ldots, F_t, G_0, \ldots, G_{t-1}$. Without loss of generality, F_i misses vertex i for $0 \leq i \leq t$, and F_t contains the edges $\{\{k, t + k + 1\} : 0 \leq k < t\}$. The union of any two near 1-factors contains a nonnegative number of even cycles and a path with an even number of edges. For $0 \leq i \leq t$, form triangles T_{ij} by adding a_i to each edge $e_{ij} \in F_i$. As in the proof of Theorem 4.5, for $0 \leq i < t$, use the edges of G_i to convert every triangle T_{ij} into a kite. Then add edge $\{i, a_i\}$ to triangle T_{ti} constructed from edge $\{i, t + 1 + i\}$. What remains is the single edge $\{t, a_t\}$ together with all edges on W.

When $w \notin \{2, 4\}$, place an $\mathscr{CN}(w, 4)$ on W of cost $\binom{w}{2}$ so that a_t appears in a triangle in the decomposition, and use the edge $\{t, a_t\}$ to convert this to a kite. We use a decomposition having $1 \leq \delta \leq 4$ triangles, therefore getting a solution with at most 3 triangles. Such a decomposition exists by Theorem 4.1 if $w \not\equiv 0, 1 \pmod 8$. If $w \equiv 0, 1 \pmod 8$ we build a solution using 4 triangles as follows. If $w \equiv 1 \pmod 8$, form an $\mathscr{CN}(w - 2, 4)$ on vertices

$\{0,\ldots,w-3\}$ with 3 triangles. Add the triangle $(w-3,w-2,w-1)$ and the 4-cycles $\{(2h,w-2,2h+1,w-1) : 0 \le h < \frac{w-3}{2}\}$. For $w = 8$ a solution with 4 triangles is given in Section 4.6.3. In general, for $w \equiv 0 \pmod 8$, form an $\mathscr{OMN}(w-8,4)$ on vertices $\{0,\ldots,w-9\}$ with 4 triangles. Add the 4-cycles $\{(2h,w-2j,2h+1,w-2j+1) : 0 \le h < \frac{w-8}{2}\}; 1 \le j \le 4$ and an $\mathscr{OMN}(8,4)$ without triangles on the 8 vertices $\{w-8,\ldots,w-1\}$.

Two values for w remain. When $w = 2$, an $\mathscr{OMN}(5,3;4,1)$ is also an $\mathscr{OMN}(5,3;4,2)$. The case when $v = 7$ and $w = 4$ is given in Section 4.6.3. The solution given has only 1 triangle.

Henceforth $w \le t$. For $t > 2$, form a near 1-factorization $\{F_0,\ldots,F_{w-1},G_0,\ldots,G_{2t-1-w}\}$ of $K_v \setminus C_t$, where C_t is the t-cycle on $(0,1,\ldots,t-1)$; such a factorization exists [169]. Name the factors so that the missing vertex in F_i is $\lfloor i/2 \rfloor$ for $0 \le i < w$ (this can be done, as every vertex i satisfying $0 \le i < t$ is the missing vertex in two of the near 1-factors). Form triangles using $F_0,\ldots,F_{w-1}$ and convert to kites using $G_0,\ldots,G_{w-1}$ as before. There remain $2(t-w)$ near 1-factors $G_w,\ldots,G_{2t-1-w}$. For $0 \le h < t-w$, $G_{w+2h} \cup G_{w+2h+1}$ contains even cycles and an even path, and so partitions into P_3s. Then the edges remaining are (1) the edges of the t-cycle; (2) the edges $\{\{\lfloor i/2 \rfloor, a_i\} : 0 \le i < w\}$; and (3) all edges on W. For $0 \le i < \lfloor \frac{w}{2} \rfloor$, form triangle (i,a_{2i},a_{2i+1}) and add edge $\{i,i+1\}$ to convert it to a kite. Edges $\{\{i,i+1 \bmod t\} : \lfloor \frac{w}{2} \rfloor \le i < t\}$ of the cycle remain from (1); edge $\{\frac{w-1}{2}, a_{w-1}\}$ remains when w is odd, and no edge remains when w is even, from (2); and all edges excepting a set of $\lfloor \frac{w}{2} \rfloor$ disjoint edges on W remain.

When $w \ne 3$, we partition the remaining edges in (1) (which form a path of length $t - \lfloor \frac{w}{2} \rfloor$), into P_3s when $t - \lfloor \frac{w}{2} \rfloor$ is even, and into P_3s and the P_2 $\{0,t-1\}$ when $t - \lfloor \frac{w}{2} \rfloor$ is odd. We adjoin edge $\{\frac{w-1}{2}, a_{w-1}\}$ to the P_3 (from the t-cycle) containing the vertex $\frac{w-1}{2}$ to form a P_4. Finally, we apply Lemma 4.2 to exhaust the remaining edges on W.

When $w = 3$, the remaining edges are those of the path $[0,t-1,t-2,\ldots,2,1,a_2]$ and edges $\{\{a_2,a_0\},\{a_2,a_1\}\}$. Include $\{\{1,2\},\{1,a_2\},\{a_2,a_0\},\{a_2,a_1\}\}$ in the decomposition, and partition the remainder into P_3s and, when $v \equiv 3 \pmod 4$, one P_2 $\{0,t-1\}$.

The case when $t = 2$ is done in Example 4.2 (the construction is exactly that given above, except that we start with a near 1-factorization of $K_5 \setminus \{\{0,1\},\{0,3\}\}$). □

4.4.2 $\mathscr{MON}(n,v;4,2)$

Theorem 4.7 *For $C' = 2$ and $v \le 2w$,*

$$\mathit{wavecost}\,\mathscr{MON}(v+w,v;4,2) = \mathit{wavecost}\,\mathscr{MON}(v+w,4).$$

Proof: It suffices to prove the statement for $v \in \{2w-2,2w-1,2w\}$. When $v = 2w-1$, apply the construction given in the proof of Theorem 4.6, where we noted that there are at most 3 triangles. The proof of Theorem 4.6 provides explicit solutions when $w \in \{2,4\}$.

Now suppose that $v = 2w$. In the proof of Theorem 4.5, $\frac{v}{2} = w$ triangles containing one edge on V and two edges between a vertex of V and a_{w-1} remain. Then convert $w-1$ triangles to kites using edges on W incident to a_{w-1}. That leaves one triangle. When the remaining edges on the $w-1$ vertices of W support a $\mathscr{MON}(w-1,4)$ that contains at most two triangles, we are done. It remains to treat the cases when $w-1 \equiv 2,7 \pmod 8$ or

$w-1=4$.For the first case, let x be one vertex of the triangle left containing a_{w-1}, namely (a_{w-1},x,y). Consider the pendant edge $\{x,t\}\in G_{w-2}$ used in a kite containing a_{w-2}. Delete $\{x,t\}$ from this kite and adjoin $\{a_{w-3},a_{w-2}\}$ to the unique triangle so formed forming another kite. Finally adjoin $\{x,t\}$ to the triangle (a_{w-1},x,y). Proceed as before, but partition all edges on $\{a_0,\ldots,a_{w-2}\}$ except edge $\{a_{w-3},a_{w-2}\}$ into 4-cycles and kites. The case when $w-1=4$ is similar, but we leave three of the triangles arising from F_{w-1} and partition $K_5\setminus P_3$ into two kites.

Now suppose that $v=2w-2$. We do a construction similar to that above. In the proof of Theorem 4.5, there remain $3\frac{v}{2}=3(w-1)$ triangles joining a_{w-3} (resp. a_{w-2},a_{w-1}) to F_{w-3} (resp. F_{w-2},F_{w-1}). Then convert the $w-1$ triangles containing a_{w-1} to kites using edges on W incident to a_{w-1}, $w-2$ triangles containing a_{w-2} to kites using the remaining edges on W incident to a_{w-2}, and $w-3$ triangles containing a_{w-3} to kites using edges on W incident to a_{w-3}. That leaves three triangles. So, if $w-3\equiv 0,1 \pmod 8$ we are done. Otherwise, as above, choose in each of the three remaining triangles vertices x_1,x_2,x_3; consider the edges $\{x_1,t_1\}$ (resp. $\{x_2,t_2\}$) appearing in the kites containing a_{w-4} and x_1 (resp. a_{w-4} and x_2), and the edge $\{x_3,t_3\}$ in the kite containing a_{w-5} and x_3. Delete these edges and adjoin them to the three remaining triangles. Finally adjoin the edges $\{a_{w-4},a_{w-5}\}$ and $\{a_{w-4},a_{w-6}\}$ to the two triangles obtained from the two kites containing a_{w-4}, and adjoin the edge $\{a_{w-5},a_{w-6}\}$ to the triangle obtained from the kite containing a_{w-5}. Proceed as before, but partition all edges on $\{a_0,\ldots,a_{w-4}\}$ except the triangle $(a_{w-6},a_{w-5},a_{w-4})$ into 4-cycles and kites. □

Theorem 4.8 *1. When $v>2w$ is even,*

$$wavecost\,\mathscr{MCN}(v+w,v;4,2)=\left\lceil\left(2\binom{v}{2}+\binom{w}{2}\right)/4\right\rceil.$$

2. When $v>2w$ is odd,

$$wavecost\,\mathscr{MCN}(v+w,v;4,2)=\left\lceil\left(2\binom{v}{2}+\frac{(w-1)(w+1)}{2}\right)/4\right\rceil.$$

Proof: First we treat the case when v is even. Then (by Theorem 4.5) an $\mathscr{CN}(v+w,v;4,2)$ must employ vw or $vw-1$ neutral edges, using all vw edges between V and W. Each such graph uses two edges on V and none on W, except that a single graph may use one on V and one on W. Now the edges of V must appear on $\lceil\frac{1}{2}\binom{v}{2}\rceil$ different wavelengths, and these wavelengths use at most one edge on W (when $v\equiv 2 \pmod 4$). Thus at least $\lceil\binom{w}{2}/4\rceil$ additional wavelengths are needed when $v\equiv 0 \pmod 4$, for a total of $\lceil\binom{v}{2}/2+\binom{w}{2}/4\rceil$. When $v\equiv 2 \pmod 4$, at least $\lceil(\binom{w}{2}-1)/4\rceil$ additional wavelengths are needed; again the total is $\lceil\binom{v}{2}/2+\binom{w}{2}/4\rceil$. Theorem 4.5 realizes this bound.

When v is odd, first suppose that w is even. In order to realize the bound of Theorem 4.6 for drop cost, by Lemma 4.1, $\frac{w}{2}$ neutral edges appear in subgraphs with one neutral edge and all other neutral edges appear in subgraphs with two. In both cases, two edges between V and W are consumed by such a subgraph. When two neutral edges are used, no edge on W can be used ; when one neutral edge is used, one edge on W can also be

used. It follows that the number of wavelengths is at least $\frac{1}{2}(\binom{v}{2} - \frac{w}{2}) + \frac{w}{2} + \frac{1}{4}(\binom{w}{2} - \frac{w}{2})$. This establishes the lower bound. The case when w is odd is similar. The proof of Theorem 4.6 gives constructions with at most 3 triangles and so establishes the upper bound except when $v \equiv 1 \pmod 4$ and $w \equiv 3 \pmod 4$, $w \neq 3$, where the construction employs one more graph than the number of wavelengths permitted. However, one graph included is the P_2 $\{0, t-1\}$, and in the decomposition on W, there is a triangle. These can be placed on the same wavelength to realize the bound. □

When $v \equiv 1 \pmod 4$ and $w \equiv 3 \pmod 4$, $w \neq 3$, we place a disconnected graph, $P_2 \cup K_3$, on one wavelength in order to meet the bound. The construction of Theorem 4.6 could be modified to avoid this by instead using a decomposition of $K_w \setminus (K_3 \cup \frac{w-3}{2}K_2)$ into 4-cycles and kites, and using the strategy used in the case for $w = 3$. In this way, one could prove the slightly stronger result that the number of (connected) subgraphs in the decomposition matches the lower bound on number of wavelengths needed.

In Theorem 4.4, the number of wavelengths and the drop cost are minimized simultaneously by the constructions given; each constructed $\mathscr{OMN}(v+w, v; 4, 1)$ has not only the minimum drop cost but also the minimum number of wavelengths over all $N(v+w, v; 4, 1)$s. This is not the case in Theorem 4.8. For example, when $v > (1 + \sqrt{2})w$, it is easy to construct an $N(v + w, v; 4, 2)$ that employs only $\lceil \binom{v}{2}/2 \rceil$ wavelengths, which is often much less than are used in Theorem 4.8. We emphasize therefore that a $\mathscr{MOMN}(v + w, v; 4, 2)$ minimizes the number of wavelengths over all $\mathscr{OMN}(v + w, v; 4, 2)$s, *not necessarily* over all $N(v + w, v; 4, 2)$s.

4.5 Case $C' = 3$

4.5.1 $\mathscr{OMN}(n, v; 4, 3)$

Theorem 4.9 *Let $v + w \geq 5$.*

1. *When $w \geq 1$, cost $\mathscr{OMN}(v + w, v; 4, 3)$ = cost $\mathscr{OMN}(v + w, 4)$.*
2. *cost $\mathscr{OMN}(v + 0, v; 4, 3)$ = cost $\mathscr{OMN}(v, 3)$.*

Proof: The second statement is trivial. Moreover *cost* $\mathscr{OMN}(n, 4)$ = *cost* $\mathscr{OMN}(n, 3)$ when $n \equiv 1, 3 \pmod 6$, and hence the first statement holds when $v + w \equiv 1, 3 \pmod 6$. To complete the proof it suffices to treat the upper bound when $w = 1$.

When $v + 1 \equiv 5 \pmod 6$, there is a maximal partial triple system $(X, \mathcal{B})$ with $|X| = v + 1$ covering all edges except those in the 4-cycle (r, x, y, z). Set $W = \{r\}$, $V = X \setminus W$, and add the 4-cycle to the decomposition to obtain an $\mathscr{OMN}(v + 1, v; 4, 3)$.

When $v \equiv 1, 5 \pmod 6$, set $\ell = v - 1$ and when $v \equiv 3 \pmod 6$ set $\ell = v - 3$. Then ℓ is even. Form a maximal partial triple system $(V, \mathcal{B})$, $|V| = v$, covering all edges except those in an ℓ-cycle $(0, 1, \ldots, \ell - 1)$ [80]. Add a vertex a_0 and form kites $(a_0, 2i, 2i + 1; (2i + 2) \bmod \ell)$ for $0 \leq i < \frac{\ell}{2}$. For $i \in \{\ell, \ldots, v - 1\}$, choose a triple $B_i \in \mathcal{B}$ so that $i \in B_i$ and $B_i = B_j$ only if $i = j$. Add $\{a_0, i\}$ to B_i to form a kite. This yields an $\mathscr{OMN}(v + 1, v; 4, 3)$. □

4.5.2 $\mathscr{MON}(n, v; 4, 3)$

We focus first on lower bounds in Section 4.5.2 and then we provide constructions attaining these lower bounds in Section 4.5.2.

Lower Bounds

When $C' = 3$, Theorem 4.9 makes no attempt to minimize the number of wavelengths. We focus on this case here. Except when $n \in \{2, 4\}$ or $v = n$, $cost\,\mathscr{ON}(n, v; 4, 3) = \binom{n}{2}$, and every graph in an $\mathscr{ON}(n, v; 4, 3)$ is a triangle, kite, or 4-cycle. Let δ, κ, and γ denote the numbers of triangles, kites, and 4-cycles in the grooming, respectively. Then $3\delta + 4\kappa + 4\gamma = \binom{n}{2}$, and the number of wavelengths is $\delta + \kappa + \gamma$. Thus in order to minimize the number of wavelengths, we must minimize the number δ of triangles. We focus on this equivalent problem henceforth.

In an $\mathscr{ON}(n, v; 4, 3)$, for $0 \le i \le 3$ and $0 \le j \le 4$, let δ_{ij}, κ_{ij}, and γ_{ij} denote the number of triangles, kites, and 4-cycles, respectively, each having i edges on V and j edges between V and W. The only counts that can be nonzero are δ_{00}, δ_{02}, δ_{12}, δ_{30}; κ_{00}, κ_{01}, κ_{02}, κ_{03}, κ_{12}, κ_{13}, κ_{22}, κ_{31}; γ_{00}, γ_{02}, γ_{04}, γ_{12}, γ_{22}. We write $\sigma_{ij} = \kappa_{ij} + \gamma_{ij}$ when we do not need to distinguish kites and 4-cycles. Our objective is to minimize $\delta_{00} + \delta_{02} + \delta_{12} + \delta_{30}$ subject to certain constraints; we adopt the strategy of [79] and treat this as a linear program.

Let $\varepsilon = 0$ when $v \equiv 1, 3 \pmod 6$, $\varepsilon = 2$ when $v \equiv 5 \pmod 6$, and $\varepsilon = \frac{v}{2}$ when $v \equiv 0 \pmod 2$. We specify the linear program in Figure 4.1. The first row lists the primal variables. The second lists coefficients of the objective function to be minimized. The remainder list the coefficients of linear inequalities, with the final column providing the *lower bound* on the linear combination specified. The first inequality states that the number of edges on V used is at least the total number on V, while the second specifies that the number of edges used between V and W is at most the total number between V and W. For the third, when $v \equiv 5 \pmod 6$ at least four edges on V are not in triangles, and so at least two graphs containing edges of V do not have a triangle on V; when $v \equiv 0 \pmod 2$ every graph can induce at most two odd degree vertices on V, yet all are odd in the decomposition.

δ_{30}	δ_{12}	δ_{02}	δ_{00}	κ_{31}	σ_{22}	κ_{13}	σ_{12}	γ_{04}	κ_{03}	σ_{02}	κ_{01}	σ_{00}	
1	1	1	1	0	0	0	0	0	0	0	0	0	
3	1	0	0	3	2	1	1	0	0	0	0	0	$\binom{v}{2}$
0	-2	-2	0	-1	-2	-3	-2	-4	-3	-2	-1	0	$-vw$
0	1	0	0	0	1	1	1	0	0	0	0	0	ε

Figure 4.1: The linear program for $\mathscr{ON}(n, v; 4, 3)$.

We do not solve this linear program. Rather we derive lower bounds by considering its dual. Let y_1, y_2, and y_3 be the dual variables. A dual feasible solution has $y_1 = \frac{1}{3}$, $y_2 = 1$, and $y_3 = \frac{4}{3}$, yielding a dual objective function value of $\frac{1}{6}v(v-1) - vw + \frac{4}{3}\varepsilon$. Recall that every dual feasible solution gives a lower bound on all primal feasible solutions

On the other hand, $3\delta \equiv \binom{n}{2} \pmod 4$ and so $\delta \equiv 9\delta \equiv 3\binom{n}{2} \pmod 4$. The value of $3\binom{n}{2} \pmod 4$ is in fact the value of t given in Theorem 4.1. Therefore if x is a lower bound on δ in an $\mathscr{ON}(n, v; 4, 3)$, so is $\langle x \rangle_n$, where $\langle x \rangle_n$ denotes the smallest nonnegative integer $\overline{x}$ such that $\overline{x} \geq x$ and $\overline{x} \equiv 3\binom{n}{2} \pmod 4$.

The discussion above proves the general lower bound on the number of triangles:

Theorem 4.10 *Let $v + w \geq 5$, and let*

$$L(v,w) = \begin{cases} \frac{1}{6}v(v-1) - vw & \text{if } v \equiv 1,3 \pmod 6 \\ \frac{1}{6}v(v-1) - vw + \frac{8}{3} & \text{if } v \equiv 5 \pmod 6 \\ \frac{1}{6}v(v+3) - vw & \text{if } v \equiv 0 \pmod 2 \end{cases}$$

Then the number of triangles in an $\mathscr{ON}(v + w, v; 4, 3)$ is at least

$$\delta_{\min}(v,w) = \langle L(v,w) \rangle_{v+w}$$

Remark 4.1 *In particular, if v is odd and $w \geq \lceil \frac{v-1}{6} \rceil$ or if v is even and $w \geq \lceil \frac{v-4}{6} \rceil$, then $L(v,w) \leq 0$ and the minimum number of triangles is $\delta_{\min}(v,w) = \langle 0 \rangle_{v+w} \leq 3$.*

Upper Bounds

We first state two simple lemmas to be used intensively in the proof of Theorem 4.11. The following result shows that in fact we do not need to check *exactly* that the number of triangles of an optimal construction meets the bound of Theorem 4.10.

Lemma 4.3 *Any $\mathscr{ON}(v + w, v; 4, 3)$ is a $\mathscr{MON}(v + w, v; 4, 3)$ if the number of triangles that it contains is at most* $\max(3, \lceil L(v,w) \rceil + 3)$.

Proof: In the closed interval $[\lceil L(v,w) \rceil, \lceil L(v,w) \rceil + 3]$ there is exactly one integer congruent to $3\binom{n}{2} \pmod 4$, and so necessarily exactly one integer equal to $\delta_{\min}(v,w)$. □

Combining Remark 4.1 and Lemma 4.3 we deduce that when v is odd and $w \geq \lceil \frac{v-1}{6} \rceil$ or if v is even and $w \geq \lceil \frac{v-4}{6} \rceil$, to prove the optimality of a construction it is enough to check that there are at most three triangles.

As a prelude to the constructions, let $(V, \mathcal{B})$ be a partial triple system, $V = \{0, \ldots, v-1\}$, and $\mathcal{B} = \{B_1, \ldots, B_b\}$. Let r_i be the number of blocks of $\mathcal{B}$ that contain $i \in V$. A *headset* is a multiset $S = \{s_1, \ldots, s_b\}$ so that $s_k \in B_k$ for $1 \leq k \leq b$, and for $0 \leq i \leq v-1$ the number of occurrences of i in S is $\lfloor \frac{r_i}{3} \rfloor$ or $\lceil \frac{r_i}{3} \rceil$.

Lemma 4.4 *Every partial triple system has a headset.*

Proof: Form a bipartite graph Γ with vertex set $V \cup \mathcal{B}$, and an edge $\{v, B\}$ for $v \in V$ and $B \in \mathcal{B}$ if and only if $v \in B$. The graph Γ admits an equitable 3-edge-colouring [89]; that is, the edges can be coloured green, white, and red so that every vertex of degree d is incident with either $\lfloor d/3 \rfloor$ or $\lceil d/3 \rceil$ edges of each colour. Then for $1 \leq k \leq b$, B_k is incident to exactly three edges, and hence to exactly one edge $\{i_k, B_k\}$ that is green; set $s_k = i_k$. Then $(s_1, \ldots, s_b)$ forms the headset. □

Theorem 4.11 *Let $v + w \geq 5$. When $w \geq 1$,*

$$wavecost\,\mathscr{MCN}(v+w, v; 4, 3) = \left\lceil \left(\binom{v+w}{2} + \delta_{\min}(v, w) \right) / 4 \right\rceil.$$

Proof: The lower bound follows from Theorem 4.10, so we focus on the upper bound.

When $w \geq 1$, an $\mathscr{CN}(v+w, v; 4, 3)$ of cost $\binom{v+w}{2}$ is an $\mathscr{CN}(v+w, v-1; 4, 3)$. Let us show that it suffices to prove the statement for $w \leq \frac{v+9}{6}$ when v is odd, and for $w \leq \frac{v+4}{6}$ when v is even. Equivalently, we show that if it is true for these values of w, then it follows for any w. Note that $\delta_{\min}(v, w) \leq 3$ if $\delta_{\min}(v+1, w-1) \leq 3$.

Indeed, let v be even. If $w = \lfloor \frac{v+4}{6} \rfloor + 1$, the result follows from the case for $v+1$ (odd) and $w - 1 = \lfloor \frac{v+4}{6} \rfloor \leq \frac{v+1+9}{6}$, in which case $\delta_{\min}(v+1, w-1) = \langle 0 \rangle_{v+w}$. If $w = \lfloor \frac{v+4}{6} \rfloor + 2$ it follows from the case for $v+1$ (odd) and $w - 1 = \lfloor \frac{v+4}{6} \rfloor + 1 \leq \frac{v+1+9}{6}$, and $\delta_{\min}(v+1, w-1) = \langle 0 \rangle_{v+w}$. If $w \geq \lfloor \frac{v+4}{6} \rfloor + 3$ it follows from the case for $v+2$ (even) and $w-2$.

Let v be odd. If $w = \lfloor \frac{v+9}{6} \rfloor + 1$ it follows from the case for $v+1$ (even) and $w-1$, which has been already proved (in this case also $\delta_{\min}(v+1, w-1) = \langle 0 \rangle_{v+w}$). If $w \geq \lfloor \frac{v+9}{6} \rfloor + 2$ it follows from the case for $v+2$ (odd) and $w-2$.

In each case, we use the same general prescription. Given a partial triple system $(V, \mathcal{B})$, a headset $S = \{s_1, \ldots, s_b\}$ is formed using Lemma 4.4. Add vertices $W = \{a_0, \ldots, a_{w-1}\}$, a set disjoint from V of size $w \geq 1$. For each i let D_i be a subset of $\{0, \ldots, w-1\}$, which is specified for each subcase, and that satisfies the following property: $|D_i|$ is at most the number of occurrences of i in the headset S. Among the blocks B_k such that $s_k = i$, we choose $|D_i|$ of them, namely the subset $\{B_k^j : j \in D_i\}$, and form $|D_i|$ kites by adding for each $j \in D_i$ the edge $\{a_j, i\}$ to the block B_k^j.

The idea behind the construction is that if we can choose $|D_i| = w$, we use all the edges between V and W leaving a minimum number of triangles in the partition of V (see Case **O1a**). Unfortunately it is not always possible to choose $|D_i| = w$, in particular when w is greater than the number of occurrences of i in the headset. So we distinguish different cases:

Case O1a. $v = 6t+1$ **or** $6t+3$ **and** $w \leq \frac{v-1}{6}$**.** Let $(V, \mathcal{B})$ be a Steiner triple system. For $0 \leq i < v$, let $D_i = \{0, \ldots, w-1\}$. Apply the general prescription. If $v = 6t+1$, i appears t times in S and $w \leq \frac{v-1}{6} = t$. If $v = 6t+3$, i appears t or $t+1$ times in S and $w \leq t$. In both cases $|D_i|$ is at most the number of occurrences of i in S, so the construction applies and all the edges between V and W are used in the kites. All the edges on V are used and $\frac{v(v-1)}{6} - vw$ triangles remain. Finally, it remains to partition the edges of W. When $w \notin \{2, 4\}$, form a $\mathscr{MCN}(w, 4)$ on W, and doing so we have at most $\delta_{\min}$ triangles. If $w = 2$ or $w = 4$ remove edges $\{a_0, 0\}$ and $\{a_1, 0\}$ from their kites and partition K_W together with these edges into a triangle ($w = 2$) or two kites ($w = 4$).

Case O1b. $v = 6t+5$ **and** $w \leq \frac{v-1}{6}$**.** Form a partial triple system $(V, \mathcal{B})$ covering all edges except those in the C_4 $(0, 1, 2, 3)$. For $0 \leq i \leq 3$, let $D_i = \{0, \ldots, w-2\}$ and for $4 \leq i < v$ $D_i = \{0, \ldots, w-1\}$. Apply the general prescription. Add the kites $(a_{w-1}, 1, 2; 3)$

and $(a_{w-1}, 3, 0; 1)$. Here again i appears at least t times in S and $w \le t$. So D_i is at most the number of occurrences of i in S. Again we have used all the edges on V and all the edges between V and W. It remains to partition the edges of W, and this can be done as in the Case **O1a**.

Case O2. $v = 6t + 3$ **and** $w = t + 1$, $v > 3$. Form a partial triple system covering all edges except those on the v-cycle $\{\{i, (i+1) \bmod v\} : 0 \le i < v\}$ [80]. Set $D_i = \{1, \ldots, w-1\}$ for all i. Apply the general prescription. Adjoin edges from a_0 to a partition of the cycle, minus edge $\{0, v-1\}$, into P_3s. The only edge between V and W that remains is $\{a_0, v-1\}$. When an $\mathscr{OMN}(w, 4)$ exists having 1, 2, 3, or 4 triangles, this edge is used to convert a triangle to a kite. This handles all cases except when $w \in \{2, 4\}$. In these cases, remove the pendant edge $\{a_1, v-1\}$ from its kite. When $w = 2$, $\{a_0, a_1, v-1\}$ forms a triangle. When $w = 4$, partition the edges on W together with $\{a_0, v-1\}$ and $\{a_1, v-1\}$ into two kites.

Case O3. $v = 6t + 1$ **and** $w = t + 1$.

When $t = 1$, a $\mathscr{MOMN}(7+2, 7; 4, 3)$ has $\mathcal{B} = \{(0, a_1, a_0; 6), (2, 0, 6; a_1), (3, 0, 4; a_1), (1, 0, 5; a_1), (3, 6, 5; a_0), (4, 6, 1; a_1), (3, 2, 1; a_0), (5, 2, 4; a_0), (a_0, 2, a_1, 3)\}$.

A solution with $t = 2$ is given in Section 4.6.4.

When $t \ge 3$, form a 3-GDD of type 6^t with groups $\{\{6p + q : 0 \le q < 6\} : 0 \le p < t\}$. Let $D_{6p+q} = \{0, \ldots, w-2\} \setminus \{p\}$ for $0 \le p < t$ and $0 \le q < 6$. Apply the general prescription. For $0 \le p < t$, on $\{6p + q : 0 \le q < 6\} \cup \{v-1\} \cup \{a_{w-1}, a_p\}$ place a $\mathscr{MOMN}(7+2, 7; 4, 3)$ obtained from the solution $\mathcal{B}$ for $t = 1$, by replacing q by $6p + q : 0 \le q < 6$, 6 by $v - 1$ a_0 by a_{w-1} and a_1 by a_p; then omit the kite $(a_p, 6p, a_{w-1}; v-1)$. All edges on W remain; the edges $\{a_{w-1}, 6p\}$ and $\{a_p, 6p\}$ remain for $0 \le p < t$, and the edge $\{a_{w-1}, v-1\}$ remains.

Add the kites $(a_{w-2}, 6(w-2), a_{w-1}; v-1)$ and for $0 \le j < w-2 = t-1$ $(6j, a_{w-1}, a_j; a_{w-2})$. If $w - 2 \notin \{2, 4\}$, that is $t \notin \{3, 5\}$, place a $\mathscr{MOMN}(w-2, 4)$ on $W - a_{w-2} - a_{w-1}$. Note that, as $3\binom{w-2}{2} \equiv 3\binom{v+w}{2} \pmod 4$, we have the right number of triangles (at most 3). If $w - 2 \in \{2, 4\}$ remove edges $\{a_0, w-2\}$ and $\{a_1, w-2\}$ from their kites, and partition K_w together with these edges.

Case O4. $v = 6t + 5$ **and** $w = t + 1$.

For $t = 0$, a $\mathscr{MOMN}(5+1, 5; 4, 3)$ has kites $(3, a_0, 0; 1)$, $(1, a_0, 2; 3)$, $(1, 3, 4; a_0)$, and triangle $(0, 2, 4)$.

For $t = 1$, let $V = \{0, \ldots, 10\}$ and $W = \{a_0, a_1\}$. A $\mathscr{MOMN}(11+2, 11; 4, 3)$ is formed by using an $\mathscr{MOMN}(5+1, 5; 4, 3)$ on $\{0, 1, 2, 3, 4\} \cup \{a_0\}$, and a partition of the remaining edges, denoted by $\mathcal{Q}$, into 15 kites and a triangle. So we have two triangles, attaining $\delta_{min}(11, 2)$ as $13 \equiv 5 \pmod 8$. The partition of $\mathcal{Q}$ is as follows: the triangle $(a_0, a_1, 10)$ and the kites $(0, 6, 5; a_0)$, $(1, 8, 6; a_0)$, $(2, 9, 7; a_0)$, $(3, 10, 8; a_0)$, $(4, 6, 9; a_0)$, $(8, 9, 0; a_1)$, $(5, 7, 1; a_1)$, $(5, 8, 2; a_1)$, $(6, 7, 3; a_1)$, $(5, 10, 4; a_1)$, $(3, 9, 5; a_1)$, $(2, 10, 6; a_1)$, $(0, 10, 7; a_1)$, $(4, 7, 8; a_1)$, and $(1, 10, 9; a_1)$.

For $t = 2$, a $\mathscr{MOMN}(17+3, 17; 4, 3)$ is given in Section 4.6.4.

For $t \geq 3$, form a 3-GDD of type 6^t with groups $\{\{6p+q : 0 \leq q < 6\} : 0 \leq p < t\}$. Let $D_{6p+q} = \{0, \ldots, w-2\} \setminus \{p\}$ for $0 \leq p < t$ and $0 \leq q < 6$. Apply the general prescription. There remain uncovered for each p the edges of the set $\mathcal{Q}_p$ obtained from the complete graph on the set of vertices $\{6p+q : 0 \leq q < 6\} \cup \{v-5, v-4, v-3, v-2, v-1\} \cup \{a_{w-1}, a_p\}$ minus the complete graph on $\{v-5, v-4, v-3, v-2, v-1\} \cup \{a_{w-1}\}$.

To deal with the edges of $\mathcal{Q}_p$, we start from a partition of $\mathcal{Q}$, where we replace pendant edges in kites as follows: replace $\{a_1, 4\}$ by $\{a_1, 10\}$, $\{a_0, 8\}$ by $\{a_0, 10\}$, and $\{a_1, 2\}$ by $\{a_0, 8\}$. We delete the triangle $(a_0, a_1, 10)$, resulting in a new partition of $\mathcal{Q}$ into 15 kites and the 3 edges $\{a_0, a_1\}$, $\{a_1, 2\}$, and $\{a_1, 4\}$. Then we obtain a partition of $\mathcal{Q}_p$ by replacing $\{0, 1, 2, 3, 4\}$ by $\{v-5, v-4, v-3, v-2, v-1\}$, $q+5$ by $6p+q$ for $0 \leq q < 6$, a_0 by a_{w-1}, and a_1 by a_p. At the end we get a partition of $\mathcal{Q}_p$ into 15 kites plus the 3 edges $\{a_{w-1}, a_p\}$, $\{a_p, v-3\}$, and $\{a_p, v-1\}$.

Now the $3t$ edges $\{\{a_{w-1}, a_p\}, \{a_p, v-3\}, \{a_p, v-1\} : 0 \leq p < t\}$ plus the uncovered edges of K_W form a K_{t+3} missing a triangle on $\{a_{w-1}, v-3, v-1\}$. If $t+3 \equiv 2, 3, 4, 5, 6, 7 \pmod 8$, use Theorem 4.1 to form a $\mathscr{ON}(t+3, 4)$ having a triangle $(v-3, v-1, a_{w-1})$ and 0, 1, or 2 other triangles; remove the triangle $(v-3, v-1, a_{w-1})$ to complete the solution with 1, 2, or 3 triangles (the triangle $(v-5, v-3, v-1)$ is still present). A variant is needed when $t+3 \equiv 0, 1 \pmod 8$. In these cases, form a $\mathscr{ON}(t+3, 4)$ (having no triangles) in which $(v-3, a_{w-1}, v-1; a_1)$ is a kite. Remove all edges of this kite, and use edge $\{a_1, v-1\}$ to convert triangle $(v-5, v-3, v-1)$ to a kite.

Finally, place a $\mathscr{MON}(5+1, 5; 4, 3)$ on $\{v-5, v-4, v-3, v-2, v-1\} \cup \{a_0\}$. Altogether we have a partition of all the edges using at most 3 triangles.

Case O5. $v = 6t+5$ **and** $w = t+2$**.**

When $t = 0$, partition all edges on $\{0, 1, 2, 3, 4\} \cup \{a_0, a_1\}$ except $\{a_0, a_1\}$ into kites $(3, 1, a_0; 0)$, $(3, 2, a_1; 0)$, $(a_1, 1, 4; 2)$, $(0, 1, 2; a_0)$, and $(3, 0, 4; a_0)$. Then a $\mathscr{MON}(5+2, 5; 4, 3)$ is obtained by removing pendant edges $\{a_0, 0\}$ and $\{a_1, 0\}$ and adding triangle $(a_0, a_1, 0)$.

When $t = 1$, a $\mathscr{MON}(11+3, 11; 4, 3)$ on $\{0, \ldots, 10\} \cup \{a_0, a_1, a_2\}$ is obtained by taking the above partition on $\{0, 1, 2, 3, 4\} \cup \{a_0, a_1\}$, the triangle (a_0, a_1, a_2), and a partition of the remaining edges (which form a graph called $\mathcal{Q}$) into 11 kites and 6 4-cycles as follows: kites $(2, 9, 7; a_0)$, $(4, 5, 10; a_0)$, $(2, 10, 6; a_1)$, $(4, 6, 9; a_2)$, $(7, 10, 0; a_2)$, $(6, 8, 1; a_2)$, $(5, 8, 2; a_2)$, $(5, 9, 3; a_2)$, $(7, 8, 4; a_2)$, $(6, 7, 5; a_2)$, and $(9, 10, 8; a_1)$; and 4-cycles $(0, 6, a_0, 5)$, $(0, 8, a_0, 9)$, $(1, 5, a_1, 7)$, $(1, 9, a_1, 10)$, $(3, 6, a_2, 7)$, and $(3, 8, a_2, 10)$.

A solution with $t = 2$ is given in Section 4.6.4.

When $t \geq 3$, form a 3-GDD of type 6^t with groups $\{\{6p+q : 0 \leq q < 6\} : 0 \leq p < t\}$. Let $D_{6p+q} = \{0, \ldots, w-3\} \setminus \{p\}$ for $0 \leq p < t$ and $0 \leq q < 6$. Apply the general prescription. Add a partition of the complete graph on $\{v-5, v-4, v-3, v-2, v-1\} \cup \{a_{w-2}, a_{w-1}\}$ as in the case when $t = 0$. It remains to partition, for each p, $0 \leq p < t$, the graph $\mathcal{Q}_p$ is obtained from the complete graph on $\{6p+q : 0 \leq q < 6\} \cup \{v-5, v-4, v-3, v-2, v-1\} \cup \{a_{w-2}, a_{w-1}, a_p\}$ minus the complete graph on $\{v-5, v-4, v-3, v-2, v-1\} \cup \{a_{w-2}, a_{w-1}\}$. This partition is obtained from that of $\mathcal{Q}$ by replacing $\{0, 1, 2, 3, 4\}$ by $\{v-5, v-4, v-3, v-2, v-1\}$, a_0 by a_{w-2}, a_1 by a_{w-1}, and a_2 by a_p. What remains is precisely the edges on W, so place a

$\mathcal{MON}(w,4)$ on W to complete the construction.

Case O6. $v = 6t+3$ **and** $w = t+2$**.**

When $t = 0$, a $\mathcal{MON}(3+2,3;4,3)$ has triangles $(a_0,0,1)$ and $\{a_1,1,2\}$ and 4-cycle $(0,2,a_0,a_1)$.

When $t = 1$, on $\{0,\ldots,8\} \cup \{a_0,a_1,a_2\}$, place kites $(2,6,4;a_0)$, $(0,8,4;a_1)$, $(0,5,7;a_1)$, $(3,6,0;a_2)$, $(1,7,4;a_2)$, $(5,8,2;a_2)$, $(1,6,5;a_2)$, $(2,7,3;a_2)$, $(3,8,1;a_2)$, $(3,5,a_0;a_2)$, $(7,a_0,6;a_2)$, $(6,8,a_1;a_2)$, $(7,a_2,8;a_0)$, and 4-cycle $(3,4,5,a_1)$. Adding the blocks of a $\mathcal{MON}(3+2,3;4,3)$ forms a $\mathcal{MON}(9+3,9;4,3)$.

A solution with $t = 2$ is given in Section 4.6.4.

When $t \geq 3$, form a 3-GDD of type 6^t with groups $\{\{6p+j : 0 \leq q < 6\} : 0 \leq p < t\}$. Let $D_{6p+q} = \{0,\ldots,w-3\} \setminus \{p\}$ for $0 \leq p < t$ and $0 \leq q < 6$. Apply the general prescription. For $0 \leq p < t$, on $\{6p+q : 0 \leq q < 6\} \cup \{v-3,v-2,v-1\} \cup \{a_{w-2},a_{w-1},a_p\}$ place a $\mathcal{MON}(9+3,9;4,3)$, omitting a $\mathcal{MON}(3+2,2;4,3)$ on $\{a_{w-2},a_{w-1},v-3,v-2,v-1\}$. Place a $\mathcal{MON}(3+2,2;4,3)$ on $\{a_{w-2},a_{w-1},v-3,v-2,v-1\}$. Remove edges $\{a_0,a_{w-2}\}$ and $\{a_1,a_{w-1}\}$ from their kites, and convert the two triangles in the $\mathcal{MON}(3+2,2;4,3)$ to kites using these. What remains is all edges on $\{a_0,\ldots,a_{w-3}\}$ and everything is in kites or 4-cycles excepting one triangle involving a_0 and one involving a_1. If $w-2 \equiv 0,1,3,6 \pmod 8$, place a $\mathcal{MON}(w-2,4)$ on $\{a_0,\ldots,a_{w-3}\}$. Otherwise partition all edges on $\{a_0,\ldots,a_{w-3}\}$ except $\{a_0,a_2\}$ and $\{a_1,a_2\}$ into kites, 4-cycles, and at most one triangle, and use the last two edges to form kites with the excess triangles involving a_0 and a_1. The partition needed is easily produced for $w-2 \in \{4,5,7,9\}$ and hence by induction for all the required orders.

Case E1. $v \equiv 0 \pmod 2$ **and** $w \leq \frac{v+2}{6}$**.** Write $v = 6t+s$ for $s \in \{0,2,4\}$. Let $L = (V,E)$ be a graph with edges

$$\{\{3i,3i+1\},\{3i,3i+2\},\{3i+1,3i+2\} : 0 \leq i < t\} \cup \{\{i,3t+i\} : 0 \leq i < 3t\},$$

together with $\{6t,6t+1\}$ when $s = 2$ and with $\{\{6t,6t+1\},\{6t,6t+2\},\{6t,6t+3\}\}$ when $s = 4$. Let $(V,\mathcal{B})$ be a partial triple system covering all edges except those in L (this is easily produced). Let $D_i = \{0,\ldots,w-2\}$ for $0 \leq i < v$. Apply the general prescription. For $0 \leq i < t$ and $j \in \{0,1,2\}$, form the 4-cycle $(a_{w-1},3i+((j+1) \bmod 3),3i+j,3t+3i+j)$. When $s = 4$, form 4-cycle $(a_{w-1},6t+2,6t,6t+3)$. When $s \in \{2,4\}$, form a triangle $(a_{w-1},6t,6t+1)$. All edges on V are used and all edges on W remain. All edges between V and W are used. Except when $w \in \{2,4\}$, or $w \equiv 2,7 \pmod 8$ and $v \equiv 2,4 \pmod 6$ form a $\mathcal{MON}(w,4)$ on W to complete the proof. When $w \equiv 2,7 \pmod 8$ and $v \equiv 2,4 \pmod 6$, convert $\{a_{w-1},6t,6t+1\}$ to a kite using an edge of the K_w, and partition the $K_w \setminus K_2$ into kites and 4-cycles. When $w \in \{2,4\}$, remove edges $\{a_0,0\}$ and $\{a_1,0\}$ from their kites, and partition K_w together with these edges.

Case E2. $v \equiv 2 \pmod 6$ **and** $w = \frac{v+4}{6}$**.** Choose m as large as possible so that $m \leq \frac{v}{2}$, $m \leq \binom{w}{2}$, and $\binom{w}{2} - m \equiv 0 \pmod 4$. Partition the $\binom{w}{2}$ edges on W into sets E_c and E_o with $|E_c| = m$, so that the edges on E_o can be partitioned into kites and 4-cycles; this is easily

done. Place these kites and 4-cycles on W. Then let $\{e_i : 0 \le i < m\}$ be the edges in E_c; let $a_{f_i} \in e_i$ when $0 \le i < m$; $f_i = 0$ when $m \le i < \frac{v-2}{2}$; and $f_{(v-2)/2} = 1$ if $m < \frac{v}{2}$. Next form a 3-GDD of type $2^{v/2}$ on V so that $\{\{2i, 2i+1\} : 0 \le i < \frac{v}{2}\}$ forms the groups, and $\mathcal{B}$ forms the blocks. For $0 \le i < \frac{v}{2}$, let $D_{2i} = D_{2i+1} = \{0, \ldots, w-1\} \setminus \{f_i\}$. Apply the general prescription. Now for $0 \le i < \frac{v}{2}$, form the triangle $(a_{f_i}, 2i, 2i+1)$ and for $0 \le i < m$ add edge e_i to form a kite. At most three triangles remain *except when* $v \in \{14, 20\}$, where four triangles remain. To treat these cases, we reduce the number of triangles; without loss of generality, the 3-GDD contains a triple $\{v-8, v-6, v-4\}$ in a kite with edge $\{a_1, v-8\}$. Remove this kite, and form kites $(a_0, v-7, v-8; v-6)$, $(a_0, v-5, v-6; v-4)$, $(a_0, v-3, v-4; v-8)$, and $(v-2, v-1, a_1; v-8)$. □

Corollary 4.1 *Let $v \ge 4$ and $\mu_3(v)$ be defined by:*

v	6	$6t, t \ge 2$	$1+6t$	$2+6t$	9	$3+6t, t \ge 2$	4	10	$4+6t, t \ge 2$	$5+6t$
$\mu_3(v)$	1	$1+t$	t	$1+t$	1	$1+t$	1	2	$2+t$	$1+t$

Then wavecost $\mathcal{MON}(v+w, v; 4, 3) = \left\lceil \frac{(v+w)(v+w-1)}{8} \right\rceil$ if and only if $w \ge \mu_3(v)$.

4.6 Some Small Constructions

4.6.1 Used in the proof of Theorem 4.2

$\mathcal{MON}(3+3, 3; 4, 1)$: $\mathcal{B} = \{(0, a_0, 1; a_2), (1, a_1, 2; a_0), (2, a_2, 0; a_1), (a_0, a_1, a_2)\}$.

$\mathcal{MON}(4+4, 4; 4, 1)$: $\mathcal{B} = \{(1, 2, a_3; a_0), (0, 3, a_2; a_1), (a_1, 1, 3; a_0), (a_0, a_2, 1; 0), (a_0, a_1, 2; 0), (a_1, a_3, 0; a_0), (2, 3, a_3, a_2)\}$.

$\mathcal{MON}(5+5, 5; 4, 1)$: $\mathcal{B} = \{(1, 2, a_3; a_0), (0, 3, a_2; a_1), (a_1, 1, 3; a_0), (a_0, a_2, 1; 0), (a_0, a_1, 2; 0), (a_1, a_3, 0; a_0), (2, a_2, 4; a_4), (3, a_3, 4), (a_2, a_3, a_4), (2, 3, a_4), (0, 4, a_0, a_4), (1, 4, a_1, a_4)\}$.

$\mathcal{MON}(6+6, 6; 4, 1)$: $\mathcal{B} = \{(1, 2, a_3; a_0), (0, 3, a_2; a_1), (a_1, 1, 3; a_0), (a_0, a_2, 1; 0), (a_0, a_1, 2; 0), (a_1, a_3, 0; a_0), (4, 5, a_5; a_4), (2, a_2, 4; a_4), (2, 3, a_4; 5), (3, 4, a_3), (a_2, a_3, a_4), (0, 4, a_0, a_4), (1, 4, a_1, a_4), (0, 5, a_0, a_5), (1, 5, a_1, a_5), (2, 5, a_2, a_5), (3, 5, a_3, a_5)\}$.

4.6.2 Used in the proof of Theorem 4.3

$\mathcal{MON}(1+2, 1; 4, 1)$: $\mathcal{B} = \{(0, a_0, a_1)\}$.

$\mathcal{MON}(2+3, 2; 4, 1)$: $\mathcal{B} = \{(0, a_0, a_1), (1, a_1, a_2), (0, 1, a_0, a_2)\}$.

$\mathcal{MON}(3+4, 3; 4, 1)$: $\mathcal{B} = \{(0, a_0, a_1), (1, a_1, a_2), (0, 1, a_0, a_2), (2, a_2, a_3), (0, 2, a_0, a_3), (1, 2, a_1, a_3)\}$.

$\mathcal{MON}(4+5, 4; 4, 1)$: $\mathcal{B} = \{(0, 1, a_0; a_3), (0, 2, a_1; a_3), (0, 3, a_2; a_3), (2, 3, a_0; a_4), (1, 3, a_1; a_4), (1, 2, a_3; 3), (0, a_3, a_4; 3), (1, a_2, a_4; 2), (a_0, a_1, a_2; 2)\}$.

4.6.3 Used in the proof of Theorem 4.6

$\mathscr{ON}(8,4)$ with 4 triangles: $\mathcal{B} = \{(1,2,0;4), (0,3,6;7), (0,7,5;2), (4,5,3;1), (1,4,7), (1,5,6), (2,3,7), (2,4,6)\}$.

$\mathscr{MON}(7+4,7;4,2)$: $\mathcal{B} = \{(a_0,4,2;3), (a_0,3,6;0), (a_0,0,5;1), (a_1,5,3;4), (a_1,4,6;1), (a_1,1,0;2), (a_2,0,4;5), (a_2,6,5;a_3), (a_2,1,2;5), (0,3,a_3;2), (1,a_0,a_2,3), (a_0,a_1,a_2,a_3), (a_1,2,6,a_3), (1,4,a_3)\}$.

4.6.4 Used in the proof of Theorem 4.11

$\mathscr{MON}(13+3,13;4,3)$: $\mathcal{B} = \{(5+i,4+i,1+i;a_1) \mid i = 0,1,\ldots,9\} \cup \{(1+i,5+i,4+i;a_0) \mid i = 10,11,12\} \cup \{(3+i,1+i,9+i;a_2) \mid i = 6,7,\ldots,12\} \cup \{(9+i,1+i,3+i;a_0) \mid i = 1,2,\ldots,5\} \cup \{(9,3,1;a_2), (0,a_1,a_2;12), (12,a_1,a_0;0), (a_0,9,a_2,10), (a_0,a_2,11;a_1), (a_0,9,a_2,10)\}$, where the sums are computed modulo 13.

$\mathscr{MON}(15+4,15;4,3)$: $\mathcal{B} = \{(1,2,3), (a_0,4,a_1,5), (a_0,10,a_1,11), (5,4,1;a_3), (7,1,6;a_1), (6,4,2;a_3), (7,5,2;a_2), (4,7,3;a_2), (6,5,3;a_3), (9,1,8;a_1), (10,1,14;a_0), (11,1,0;a_2), (13,1,12;a_2), (10,2,8;a_2), (11,2,9;a_0), (12,2,14;a_2), (0,2,13;a_3), (8,3,11;a_3), (10,3,12;a_0), (13,3,9;a_2), (14,3,0;a_3), (12,8,4;a_2), (11,4,13;a_0), (0,10,4;a_3), (9,4,14;a_1), (8,5,13;a_2), (0,5,12;a_3), (14,11,5;a_3), (10,9,5;a_2), (8,6,0;a_0), (14,13,6;a_3), (9,6,12;a_1), (10,6,11;a_2), (14,8,7;a_3), (9,7,0;a_1), (10,7,13;a_1), (12,11,7;a_0), (6,a_2,a_0;2), (7,a_2,a_1;3), (10,a_3,a_2;1), (1,a_0,a_1;2), (9,a_1,a_3;14), (8,a_3,a_0;3)\}$.

$\mathscr{MON}(17+3,17;4,3)$: $\mathcal{B} = \{(7,16,0), (a_0,a_2,0), (a_0,1,2;3), (a_0,3,4;1), (4,5,2;a_1), (1,3,5;a_0), (16,a_0,a_1;a_2), (6,10,1;a_1), (9,14,1;a_2), (15,1,7;a_2), (1,8,12;a_2), (1,0,13;a_2), (1,16,11;a_1), (2,11,6;a_1), (2,16,8;a_2), (10,15,2;a_2), (9,2,13;a_1), (0,2,12;a_1), (2,7,14;a_2), (6,13,3;a_1), (11,3,7;a_1), (12,3,16;a_2), (9,0,3;a_2), (3,10,14;a_1), (8,3,15;a_1), (14,6,4;a_2), (4,11,15;a_2), (7,12,4;a_1), (13,4,8;a_1), (4,16,9;a_2), (0,4,10;a_1), (5,12,6;a_2), (7,13,5;a_2), (8,14,5;a_1), (15,5,9;a_1), (5,16,10;a_2), (5,0,11;a_2), (9,7,6;a_0), (10,8,7;a_0), (11,9,8;a_0), (12,10,9;a_0), (13,11,10;a_0), (14,12,11;a_0), (15,13,12;a_0), (16,14,13;a_0), (0,15,14;a_0), (6,16,15;a_0), (8,6,0;a_1)\}$.

$\mathscr{MON}(17+4,17;4,3)$: $\mathcal{B} = \{(2,9,11), (9,12,16), (a_0,13,14;15), (a_0,15,16;13), (16,0,14;a_1), (13,15,0;a_0), (13,2,1;a_3), (13,12,3;a_3), (13,11,4;a_3), (5,10,13;a_1), (6,9,13;a_2), (7,8,13;a_3), (14,4,2;a_3), (14,12,5;a_3), (11,14,6;a_3), (14,10,7;a_3), (1,3,14;a_2), (9,8,14;a_3), (1,4,15;a_1), (3,5,15;a_2), (2,6,15;a_3), (15,7,12;a_3), (15,11,8;a_3), (1,16,5;a_1), (6,4,16;a_2), (3,7,16;a_3), (2,8,16;a_1), (10,16,11;a_3), (1,6,0;a_1), (4,8,0;a_2), (10,15,9,;a_3), (2,10,0;a_3), (5,0,7;a_1), (3,0,9;a_1), (12,0,11;a_1), (1,a_0,7;6), (8,6,a_0;a_3), (9,a_0,5;11), (10,a_0,4;9), (11,a_0,3;10), (2,a_0,12;8), (8,a_1,1;11), (10,a_1,6;3), (12,4,a_1;a_3), (3,a_1,2;7), (1,a_2,9;7), (10,a_2,8;3), (11,a_2,7;4), (12,a_2,6;5), (2,a_2,5;8), (3,a_2,4;5), (a_1,a_0,a_2;a_3), (12,1,10;a_3)\}$.

4.7 Conclusions

The determination of $cost\mathscr{O}\mathscr{N}(n, v; C, C')$ appears to be easier when $C' = 4$ than the case for $C' = 3$ settled in [78, 79]. Nevertheless the very flexibility in choosing kites, 4-cycles, or triangles also results in a wide range of numbers of wavelengths among decompositions with optimal drop cost. This leads naturally to the question of minimizing the drop cost and the number of wavelengths simultaneously. In many cases, the minima for both can be realized by a single decomposition. However, it may happen that the two minimization criteria compete. Therefore we have determined the minimum number of wavelengths among all decompositions of lowest drop cost for the specified values of n, v, and C'.

Part III

Degree-constrained Subgraphs

III.1 Motivation

We begin with an example. Let G be a 4-regular (multi)graph. Does G contain a 3-regular subgraph? Not necessarily, as the example of a triangle with two edges between each pair of vertices shows. Now suppose we add an arbitrary edge to G. It turns out that the claim is now true.

Theorem III.1 (Alon, Friedland, and Kalai [31]) *Every 4-regular graph plus an edge contains a 3-regular subgraph.*

Theorem III.1 was proved using facts from number theory. Using more sophisticated arguments, the same authors proved the following strengthening of Theorem III.1.

Theorem III.2 (Alon, Friedland, and Kalai [32]) *Suppose that every vertex of a graph G has degree k or $k+1$, and at least one vertex has degree $k+1$. Then for every prime power q such that $k \geq 2(q-1)$, G contains a q-regular subgraph.*

The above examples illustrate how apparently simple but difficult may be to assure the existence of a subgraph satisfying certain degree constraints. A general instance of a Degree-constrained Subgraph problem consists of an edge-weighted or vertex-weighted graph G and the objective is to find an optimal weighted subgraph, subject to certain degree constraints on the vertices of the subgraph. This class of combinatorial problems has been extensively studied due to its numerous applications in network design, as it is the case of the Minimum-degree Spanning Tree [128] and the Minimum-degree Steiner Tree [129] problems. If the input graph is bipartite, they have as particular cases classical transportation and assignment problems in operations research [83]. These problems have attracted a lot of attention in the last decades and have resulted in a large body of literature [30, 37, 53, 59, 60, 64, 109, 110, 110, 128–130, 150, 159, 161, 164, 171, 183].

In the sequel of the second part of this thesis we shall convince the reader that this family of problems provides a rich source of nice and interesting problems to apply a variety of algorithmic techniques and graph-theoretical approaches.

Relation to traffic grooming. We now discuss how degree-constrained subgraph problems are related to traffic grooming. More precisely, how the study of traffic grooming lead the author to the study of degree-constrained subgraph problems.

We have presented in Section 1.4 (page 43) an algorithm that computes a solution to the Ring Traffic Grooming problem with an approximation ratio of $O(n^{1/3} \cdot \log^2 n)$, where n is the size of the ring. This approximation ratio can be informally factorized as follows.

- a factor $\log n$ comes from the fact of partitioning the request set into $\log n$ classes of similar length;
- another factor $\log n$ comes from the fact of greedily removing subgraphs from the request graph[1];

[1] It is usual to *pay* a factor $\log n$ in greedy algorithms for NP-hard problems, the archetypical example being the $\log n$-approximation for the Minimum Set Cover problem [191].

- finally, the factor $n^{1/3}$ appears because we used the algorithm of [112] for the DENSE k-SUBGRAPH (DkS) problem (see page 20) to find dense subgraphs.

That is, if we had an approximation algorithm with ratio β for the DkS problem, this would automatically yield an approximation algorithm with ratio $O(\beta \cdot \log^2 n)$ for the RING TRAFFIC GROOMING problem.

Unfortunately, the DkS problem is a really difficult problem. First of all, note that the NP-hardness of DkS easily follows from the NP-hardness of MAXIMUM CLIQUE. On the other hand, if we do not fix the size of the subset of vertices S, then finding a densest subgraph of G reduces to an instance of the MAX-FLOW MIN-CUT problem, and hence it can be solved in polynomial time (see [156, Chapter 4] for more details). The DkS problem has attracted a lot of attention during the last decade, primarily in approximation algorithms [90, 112, 152]. Understanding the complexity of DkS remains widely open, as the gap between the best hardness result (APX-hardness [152]) and the best approximation algorithm (with ratio $O(n^{1/3-\varepsilon})$ [112]) is huge. Andersen and Chellapilla have recently studied in [36] two relaxed versions of the DkS problem. Namely, they have proved that the problem of finding a densest subgraph with *at least* k vertices can be efficiently approximated (they provide a 3-approximation), but that the problem of finding a densest subgraph with *at most* k vertices is essentially as hard as the DkS problem itself.

Therefore, it seems natural to ask whether one can efficiently find *dense* subgraphs using another approach. As it seems to be very hard to find subgraphs with the greatest density, or equivalently with the greatest *average degree*, one could try to find small subgraphs with prescribed *minimum degree*, which would also guarantee high density. This discussion naturally leads to the definition of the following problem (see Chapter 6 for more details).

$\leq k$-SIZE SUBGRAPH OF MINIMUM DEGREE $\geq d$ (kSMDd)
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Question: Does there exist a subset $S \subseteq V$, with $\lvert S \rvert \leq k$, such that $G[S]$ has minimum degree at least d?

Let us see how kSMDd is related to DkS. Suppose we are looking for an induced subgraph $G[S]$ of size at most k and with density at least ρ. In addition, assume that S is minimal, i.e., no subset of S has density greater than $\rho(S)$. This implies that every vertex of S has degree at least $\rho/2$ in $G[S]$. To see this, observe that if there is a vertex v with degree strictly smaller than $\rho/2$, then removing v from S results in a subgraph of density greater than $\rho(S)$ and of smaller size, contradicting the minimality of S. Secondly, if we have an induced subgraph $G[S]$ of minimum degree at least ρ, then S is a subset of density at least $\rho/2$. These two observations together show that, modulo a constant factor, looking for a densest subgraph of G of size at most k is equivalent to looking for the largest possible value of ρ for which kSMDρ returns YES for the parameter k. As the local degree conditions are more rigid than the global density of a subgraph, it may be more convenient to work directly with kSMDd. A better understanding of the kSMDd problem could provide an alternative way to approach the DkS problem, and therefore by transitivity also to approach the TRAFFIC GROOMING problem.

III.2 State-of-the-art and our Contribution

In the preceding section we have introduced the problem of finding a minimum subgraph with given minimum degree. It is natural to consider its dual version: finding a maximum subgraph (in terms of number of edges or vertices) with given maximum degree. These two problems and some variants are the main topic of Chapters 5, 6, and 7.

III.2.1 The role of connectivity

As mentioned above, we will study in the forthcoming chapters the problem of finding a maximum subgraph with given maximum degree. If we do not impose any further restriction on the desired subgraph, it turns out that the problem (for both the vertex and edge version) is solvable in polynomial time using matching techniques [83, 132, 158]. In fact, also the version where nodes or edges have associated weights and the objective is to maximize the total weight of a subgraph respecting the degree constraints, is solvable in polynomial time [132] (as it happens for the classical Maximum Matching problem). Even a more general version where the input contains an interval of allowed degrees for each node is known to be solvable in polynomial time [158].

Suppose now that we restrict the output subgraph to be connected, i.e., we would like to find a maximum *connected* subgraph with given maximum degree. Then the problem is exactly one of the classical NP-hard problems of Garey and Johnson's book: [132, Problem GT26]. Is this phenomenon surprising? Why does connectivity change completely the complexity of the problem? The explanation is the following: the degree of a vertex in a solution is a *local* property, in the sense that it depends only on how many of its incident edges are included in the solution; on the other hand, connectivity is a *global* property, as in order to check that a subgraph is connected one needs to explore the whole subgraph. Therefore, it seems natural that if the only constraints on the solution are the degree of the vertices, an algorithm along the lines of the classical augmenting paths algorithm to find a maximum matching (see for instance [158]) could find an optimal solution in polynomial time. This change from *local* to *global* is what makes the problem become NP-hard.

We prove in Chapter 5 that the problem is in fact even harder: it is not in Apx (see page 17) for any fixed maximum degree $d \geq 2$ unless P≠NP, and if there is a polynomial time algorithm for $d \geq 2$, with performance ratio $2^{O(\sqrt{\log n})}$, then NP $\subseteq$ DTIME($2^{O(\log^5 n)}$). These results were previously known only for the case $d = 2$ [150], that corresponds to the Longest Path problem. On the other hand, we give an approximation algorithm for general unweighted graphs with ratio $\min\{m/\log n,\ nd/(2\log n)\}$, and an approximation algorithm for general weighted graphs with ratio $\min\{n/2,\ m/d\}$. (Here $n = |V(G)|$ and $m = |E(G)|$, where G is the input graph.) The first algorithm uses an algorithm introduced in [34], which is based on the *color-coding* method. We also present a constant-factor approximation when the input graph has a low-degree spanning tree, in terms of the integer d.

We also prove in Chapter 5 that the problem of finding a minimum subgraph of given minimum degree is not in Apx for any fixed degree $d \geq 3$. These are the first hardness results for this problem.

To prove the hardness results for the two problems mentioned above, we use the error amplification technique, which is a powerful tool to prove that certain approximation algorithms cannot exist provided, as usual, that P≠NP.

III.2.2 Parameterizing the input

In order to better understand the complexity of NP-hard problems, the theory of parameterized complexity (see page 18) provides a framework to refine the measure of the complexity of such problems, by means of introducing a parameter to the input (which is commonly assumed to be small compared to the size of the input) and analyzing the running time of the algorithms in terms of that parameter. Since the degree-constrained subgraph problems we consider in this part of the thesis seem to be really hard to solve – in view of the inapproximability results of Chapter 5 –, the next natural step is to analyze their parameterized complexity. This is the topic of Chapter 6.

This approach brings us a very interesting insight. Indeed, when parameterizing by the size of the desired subgraph, the behavior of the two aforementioned problems is completely different, whereas the seem to have similar (classical) complexity. Namely, the problem of finding a maximum connected subgraph with bounded maximum degree is FPT, and the problem of finding a minimum subgraph (that of course, we can assume to be connected) with bounded minimum degree is $W[1]$-hard. The fact that the first problem is FPT follows directly from the fact that the corresponding parameter is minor closed (see page 19 for the definition of minor closed parameters), as it is proved in Lemma 7.2 (page 165). Proving the $W[1]$-hardness of the second problem is a more difficult task; see Chapter 6 for the details. Moreover, we prove in Chapter 6 that the problem of finding a minimum regular subgraph (induced or not) is $W[1]$-hard, by a reduction from the Multi-Color Clique problem (see Theorems 6.1 and 6.2 in Chapter 6).

III.2.3 Topologically restricting the input

A class of graphs is called *sparse* if for every graph G in this class, $|E(G)| = O(|V(G)|)$. Examples of sparse graph classes are planar graphs (for which the number of edges is at most three times the number of edges [95]), graphs embeddable in a fixed surface or, more generally, families of graphs excluding a fixed graph as a minor. When restricting the input graphs to belong to some sparse class, it is usual that *hard* problems become *easier*. This paradigm is the cornerstone idea of the second part of Chapter 6 and Chapters 7 and 8.

Namely, in Section 6.3 we provide explicit FPT algorithms for the discussed problems in graphs with bounded local treewidth and graphs with excluded minors. These algorithms are based on dynamic programming techniques and structural results from graph minors theory. We stress that the *classification result* asserting that these problems are FPT for graphs with excluded minors is a consequence of the powerful meta-theorems of Frick and Grohe [126] and of Dawar, Grohe, and Kreutzer [87] (see page 148 of Section 6.1.3 for the details). Our contribution is to give explicit algorithms that run considerably faster than those implied by the mentioned meta-theorems.

As mentioned in Section III.2.2, the fact that the problem of finding a maximum connected degree-bounded subgraph is FPT is a direct consequence of the Graph Minors Theorem. In general graphs, one cannot hope the function $f(k)$ to be subexponential (i.e., $2^{o(k)}$), because this would contradict the widely believed exponential time hypothesis [99]. However, when we restrict the input to be planar, then such algorithms are indeed possible. In Chapter 7 we obtain subexponential parameterized algorithms for the mentioned problem, as well as for a few variants. Our approach follows the general framework that has been developed during the last years, which combines bidimensionality theory and refined dynamic programming techniques [93, 96–100, 140]. As it is usual in bidimensionality theory, we obtain algorithms with $f(k) = 2^{O(\sqrt{k})}$. Loosely speaking, the idea of this approach is as follows. Suppose we have a graph G with branchwidth $\mathbf{bw}(G)$ and we want to decide whether a parameter $\mathbf{P}$ is at least k in G (for instance, "does G have a path of length at least k?"). We distinguish two cases according to $\mathbf{bw}(G)$:

- If $\mathbf{bw}(G)$ is *big* (greater than $\sqrt{k}$), we must exhibit a *certificate* that $\mathbf{P}(G)$ is also *big*, by proving that the parameter is minor closed and looking at its behavior on the square grid. Then the answer to the parameterized problem is automatically YES.
- Otherwise, if $\mathbf{bw}(G)$ is *small* (smaller than $\sqrt{k}$), we compute $\mathbf{P}(G)$ efficiently using Catalan structures and dynamic programming techniques over an optimal branch decomposition of G.

Finally, we deal in Chapter 8 with graphs embedded in surfaces [163]. In this case we consider a more general family of problems than the ones considered so far, as specified in the paragraph below. It is a common approach for solving NP-hard problems to use dynamic programming algorithms over a branch (or tree) decomposition of the input graph [57]. We focus on this approach, and particularize it when the input graph is embedded in a surface.

Namely, we provide in Chapter 8 a framework for the design of $2^{O(k)} \cdot n$ step dynamic programming algorithms for surface-embedded graphs on n vertices of branchwidth at most k. Our technique applies to graph problems for which dynamic programming uses tables encoding set partitions. For general graphs, the best known algorithms for such problems run in $2^{O(k \cdot \log k)} \cdot n$ steps. That way, we considerably extend the class of problems that can be solved by algorithms whose running times have a *single exponential dependence* on branchwidth, and improve the running time of several existing algorithms. Our approach is based on a new type of branch decomposition called *surface cut decomposition*, which generalizes sphere cut decompositions for planar graphs (see page 168), and where dynamic programming should be applied for each particular problem. The construction of such a decomposition uses a new graph-topological tool called *polyhedral decomposition*. The main result of this chapter – namely, Theorem 8.4 in page 208 – is that if dynamic programming is applied on surface cut decompositions, then the time dependence on branchwidth is *single exponential*. This fact is proved by a detailed analysis of how non-crossing partitions are arranged on surfaces with boundary and uses diverse techniques from topological graph theory and analytic combinatorics. As a consequence of our results, we can derive subexponential exact and parameterized algorithms in graphs of bounded genus for the problems in the mentioned category.

Chapter 5

Hardness and Approximation

An instance of a Degree-Constrained Subgraph problem consists of an edge-weighted or vertex-weighted graph $G = (V, E)$, $|V| = n$, $|E| = m$, and the objective is to find an optimal weighted subgraph, subject to certain degree constraints on the vertices of the subgraph. This chapter considers three natural Degree-Constrained Subgraph problems and studies their approximability.

The Maximum Degree-Bounded Connected Subgraph (MDBCS_d) problem takes as additional input a weight function $\omega : E \to \mathbb{R}^+$ and an integer $d \geq 2$, and asks for a subset $E' \subseteq E$ such that the subgraph $G' = (V, E')$ is connected, has maximum degree at most d, and $\sum_{e \in E'} \omega(e)$ is maximized. We prove that MDBCS_d is not in Apx for any $d \geq 2$, and that if there is a polynomial time algorithm for MDBCS_d, $d \geq 2$, with performance ratio $2^{O(\sqrt{\log n})}$, then NP $\subseteq$ DTIME($2^{O(\log^5 n)}$). On the other hand, we provide a $(\min\{m/\log n,\ nd/(2\log n)\})$-approximation algorithm for unweighted graphs, and a $(\min\{n/2,\ m/d\})$-approximation algorithm for weighted graphs. We also prove that when G has a low-degree spanning tree, the MDBCS_d problem can be approximated within a small constant factor in unweighted graphs.

The Minimum Subgraph of Minimum Degree$_{\geq d}$ (MSMD_d) problem involves finding a smallest subgraph of G with minimum degree at least d. We prove that MSMD_d is not in Apx for any $d \geq 3$ and provide an $O(n/\log n)$-approximation algorithm for the classes of graphs excluding a fixed graph as a minor, using dynamic programming techniques and a known structural result on graph minors. In particular, this approximation algorithm applies to planar graphs and graphs of bounded genus.

The Dual Degree-Dense k-Subgraph (DDDkS) problem involves finding a subgraph H of G such that $|V(H)| \leq k$ and δ_H is maximized, where δ_H is the minimum degree in H. We present a deterministic $O(n^\delta)$-approximation algorithm in general graphs, for some universal constant $\delta < 1/3$.

Keywords: approximation algorithms, degree-constrained subgraphs, hardness of approximation, Apx, PTAS, dense subgraphs, graph minors, excluded minor.

5.1 Introduction

In this chapter we consider three natural Degree-Constrained Subgraph problems and study them in terms of approximation algorithms. A general instance of a Degree-Constrained Subgraph problem [30, 37, 183] consists of an edge-weighted or vertex-weighted graph and the objective is to find an optimal weighted subgraph, subject to certain degree constraints on the vertices of the subgraph.

Degree-Constrained Subgraph problems have attracted a lot of attention in the last decades and have resulted in a large body of literature [30, 37, 110, 128–130, 150, 159, 171, 183]. The most well-studied ones are probably the Minimum-Degree Spanning Tree [128] and the Minimum-Degree Steiner Tree [129] problems.

Beyond the esthetic and theoretical appeal of Degree-Constrained Subgraph problems, the reasons for such intensive study are rooted in their wide applicability in the areas of interconnection networks and routing algorithms, among others. For instance, given an interconnection network modeled by an undirected graph, one may be interested in finding a small subset of nodes having a high degree of connectivity for each node. This translates into finding a small subgraph with a lower bound on the degree of its vertices, i.e., to the MSMD_d problem, to be defined shortly. Note that if the input graph is bipartite, these problems are equivalent to classical transportation and assignment problems in operation research.

The first problem studied in the chapter is a classical NP-hard problem listed in [132] (c.f. Problem [GT26] for the unweighted version):

Maximum Degree-Bounded Connected Subgraph (MDBCS_d)
Input: A graph $G = (V, E)$, a weight function $\omega : E \to \mathbb{R}^+$ and an integer $d \geq 2$.
Output: A subset $E' \subseteq E$ such that the subgraph $G' = (V, E')$ is connected (except, possibly, for isolated vertices), has maximum degree at most d, and $\sum_{e \in E'} \omega(e)$ is maximized.

For $d = 2$, the unweighted MDBCS_d problem corresponds to the Longest Path problem. Indeed, given the input graph G (which can be assumed to be connected), let P and G' be optimal solutions of Longest Path and MDBCS_2 in G, respectively. Then observe that $|E(G')| = |E(P)|$ unless G is Hamiltonian, in which case $|E(G')| = |E(P)| + 1$. One could also ask the question: what happens when G' is not required to be connected in the definition of MDBCS_d? It turns out that without the connectivity constraint, both the edge version and the vertex version (where the goal is to maximize the total weight of the vertices of a subgraph respecting the degree constraints) of the MDBCS_d problem are known to be solvable in polynomial time using matching techniques [83, 132, 158]. In fact, without connectivity constraints, even a more general version where the input contains an interval of allowed degrees for each node is known to be solvable in polynomial time.

The most general version of Degree-Constrained Subgraph problem is to find a subgraph under constraints given by lower and upper bounds on the degree of each vertex, the objective being to minimize or maximize some parameter (usually the size of the subgraph). A common variant ignores the lower bound on the degree and just requires

the vertices of the subgraphs to have a given maximum degree [171], in which case the typical optimization criterion is to maximize the size of a subgraph satisfying the degree constraints. The resulting problem is also called an Upper Degree-Constrained Subgraph problem in [130]. In contrast, we are unaware of existing results considering just a lower bound on the degrees of the vertices of the subgraph, except for combinatorial conditions on the existence of such a subgraph [109, 110]. In an attempt to fill this void in the literature, the last two problems considered in this chapter aim at minimizing the size of a subgraph and maximizing the lower bound on the minimum degree, respectively. For a graph H, let δ_H denote the minimum degree of the vertices in H.

Minimum Subgraph of Minimum Degree$_{\geq d}$ (MSMD$_d$)
Input: An undirected graph $G = (V, E)$ and an integer $d \geq 2$.
Output: A subset $S \subseteq V$ such that for $H = G[S]$, $\delta_H \geq d$ and $\lvert S \rvert$ is minimized.

Dual Degree-Dense k-Subgraph (DDDkS)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Output: An induced subgraph H of size $\lvert V(H) \rvert \leq k$, such that δ_H is maximized.

Note that the problem kSMDd defined in Section III.1 (page 118) is the parameterized version of the MSMD$_d$ problem defined above. MSMD$_d$ is closely related to MDBCS$_d$. Indeed, MSMD$_d$ corresponds exactly to the dual (unweighted) node-minimization version of MDBCS$_d$. MSMD$_d$ is also a generalization of the Girth problem (finding a shortest cycle), which corresponds exactly to the case $d = 2$. The MSMD$_d$ is studied in the realm of parameterized complexity in Chapter 6, where it is shown that MSMD$_d$ is W[1]-hard for $d \geq 4$ and explicit *fixed-parameter tractable* (FPT) algorithms are given for the class of graphs excluding a fixed graph as a minor and graphs of bounded local treewidth, for $d \geq 3$. Besides the above discussion, our main motivation for studying MSMD$_d$ is its close relation to the well studied Dense k-Subgraph (DkS) [112, 152] and Traffic Grooming [J2] problems. See Section III.1 (page 117) for more details.

The above discussion illustrates that the study of the above mentioned problems is very natural and that the results obtained for them can reverberate in several other important optimization problems, coming from both theoretical and practical domains.

Our Contribution: In this chapter we obtain both approximation algorithms and results on hardness of approximation. All of our hardness results are based on the hypothesis that P $\neq$ NP. More precisely, our results are the following:

- We prove that the MDBCS$_d$ problem is not in Apx for any $d \geq 2$, and that if there is a polynomial time algorithm for MDBCS$_d$, $d \geq 2$, with performance ratio $2^{O(\sqrt{\log n})}$, then NP $\subseteq$ DTIME($2^{O(\log^5 n)}$). On the other hand, we give an approximation algorithm for general unweighted graphs with ratio $\min\{m/\log n,\ nd/(2\log n)\}$, and an approximation algorithm for general weighted graphs with ratio $\min\{n/2,\ m/d\}$. The first algorithm uses an algorithm introduced in [34], which is based on the *color-coding* method. We also present a constant-factor approximation when the input graph has a low-degree spanning tree, in terms of the integer d.

- We prove that the MSMD$_d$ problem is not in Apx for all $d \geq 3$. The proof is obtained by the following two steps. First, by a reduction from Vertex Cover, we prove that MSMD$_d$ does not admit a PTAS. In particular, this implies that MSMD$_d$ is NP-hard for any $d \geq 3$. Then, we use the *error amplification* technique to prove that MSMD$_d$ is not in Apx for any $d \geq 3$. On the positive side, we give an $O(n/\log n)$-approximation algorithm for the class of graphs excluding a fixed graph H as a minor, using a known structural result on graph minors and dynamic programming over graphs of bounded treewidth. In particular, this gives an $O(n/\log n)$-approximation algorithm for planar graphs and graphs of bounded genus.
- We give a deterministic $O(n^{\delta})$-approximation algorithm for the DDDkS problem in general graphs, for some universal constant $\delta < 1/3$. We also provide a randomized $O(\sqrt{n}\log n)$-approximation algorithm, which is completely different in nature. Although the approximation ratio is significantly worse, the idea of the proof is quite simple and nice.

Organization of the chapter: The basic definitions required in this chapter can be found in Section I.2 (page 17). In Section 5.2 we establish that MDBCS$_d$ is not in Apx for any $d \geq 2$, and in Section 5.3 we present two approximation algorithms for unweighted and weighted general graphs, respectively. The constant-factor approximation for MDBCS$_d$ when the input graph has a low-degree spanning tree is provided in Section 5.3.2 for unweighted graphs. In Section 5.4 we prove that MSMD$_d$ is not in Apx for any $d \geq 3$, and in Section 5.5 we give an $O(n/\log n)$-approximation algorithm for the class of graphs excluding a fixed graph H as a minor. In Section 5.6 we give two approximation algorithms for the DDDkS problem. Finally, we conclude with some remarks and open problems in Section 5.7.

5.2 Hardness of Approximating MDBCS$_d$

As mentioned in Section 7.1, MDBCS$_2$ corresponds to the Longest Path (or Cycle) problem, which is known to not admit any constant-factor approximation [150], unless P = NP. In this section we extend this result and prove that, under the assumption that P≠NP, MDBCS$_d$ is not in Apx for any $d \geq 2$, proving first that MDBCS$_d$ is not in PTAS for any $d \geq 2$. Finally, we also prove in Theorem 5.4 that if there is a polynomial time algorithm for MDBCS$_d$, $d \geq 2$, with a performance ratio of $2^{O(\sqrt{\log n})}$, then NP $\subseteq$ DTIME($2^{O(\log^5 n)}$).

Theorem 5.1 MDBCS$_d$ *does not admit a* PTAS *for any* $d \geq 2$*, unless* P = NP*.*

Proof: We prove the result for the case when $d \geq 3$; the result for $d = 2$ follows from [150]. We give our reduction from the *traveling salesman problem* TSP(1, 2), which does not admit a PTAS unless P = NP [167]. An instance of TSP(1, 2) consists of a complete graph $G = (V, E)$ on n vertices and a weight function $f : E \to \{1, 2\}$ on its edges, and the objective is to find a traveling salesman tour of minimum weight in G.

We show that if there is a PTAS for MDBCS$_d$, $d \geq 3$, then one can construct a PTAS for TSP$(1,2)$. Towards this, we transform the graph G into a new augmented graph G' with a modified weight function g on its edges. For every vertex $v \in V$ we add to G' $d-2$ new vertices $\{v_1, \cdots, v_{d-2}\}$ and an edge from v to every vertex v_i, $1 \leq i \leq d-2$. Thus $G' = (V \cup V', E \cup E')$ where $V' = \bigcup_{v \in V}\{v_1, \cdots, v_{d-2}\}$ is the set of new vertices and $E' = \{\{v_i, v\} \mid 1 \leq i \leq d-2, v \in V\}$ is the set of new edges. We define the weight function g on the edges of G' as:

$$g(e) = \begin{cases} 3 - f(e), & e \in E, \quad \text{(weights of original edges get flipped)} \\ 3, & e \in E'. \end{cases}$$

Next we prove a claim characterizing the structure of the *maximal* solutions of MDBCS$_d$ in G'. Essentially, it shows that any given solution G_1 of MDBCS$_d$ in G' with value W can be transformed into another solution G_2 of MDBCS$_d$ in G' with value at least W, such that G_2 contains all the newly added edges and induces a Hamiltonian cycle in G.

Claim 5.1 *Any given solution $G_1 = (V \cup V', E_1)$ to* MDBCS$_d$ *in G' can be transformed in polynomial time into a solution $G_2 = (V \cup V', E_2)$ of* MDBCS$_d$ *in G' such that (i) $G_3 = (V, E \cap E_2)$ is a Hamiltonian cycle in G, and (ii) $\sum_{e \in E_2} g(e) \geq \sum_{e' \in E_1} g(e')$.*

Proof: We prove the claim by describing a series of transformations, applied in order of appearance, successively improving the solution, and eventually yielding the desired G_2. For a given edge set F, let $X(F)$ be the set of vertices containing the end-points of the edges in F.

(a) Suppose $E_1 \cap E' = \emptyset$. Then $H = (X(E_1), E_1)$ is connected and every vertex $v \in X(E_1)$ has degree at most d in H. This implies that H contains a cycle, so removing any edge from this cycle will not break connectivity. So we can remove any edge $\{u, v\}$ from this cycle and add the edges $\{u_1, u\}$ and $\{v_1, v\}$, obtaining a solution of larger weight. Therefore, we assume henceforth that $E_1 \cap E' \neq \emptyset$.

(b) Suppose $V \setminus X(E_1) \neq \emptyset$, that is, there is a vertex $v \in V$ which is not contained in $X(E_1)$. In this case, by Observation (a) there exists a vertex $u \in X(E_1)$ such that one of the edges $\{u_i, u\}$, $1 \leq i \leq d-2$, is in E_1. We then set $E_1 \leftarrow E_1 - \{\{u_i, u\}\} \cup \{\{u, v\}, \{v, v_i\} \mid 1 \leq i \leq d-2\}$. Clearly, connectivity is maintained (as removing edges from E' does not break connectivity) and the weight of solution increases by at least 1. This procedure is repeated until the current solution contains all the vertices of G.

(c) Suppose $H' = (V, E \cap E_1)$ is neither a spanning tree nor a Hamiltonian cycle. Notice that H' is connected, as removing degree 1 vertices of V' does not break connectivity. This implies that there is a cycle C in H' and a vertex v on it such that $deg_{H'}(v) \geq 3$ (otherwise, H' would be disconnected). This implies that there exists an edge $e = \{v, v_i\}$ such that $e \notin E_1$. Let $\{u, v\}$ be an edge on C. We then set $E_1 \leftarrow E_1 - \{\{u, v\}\} \cup \{\{v, v_i\}\}$. Again, connectivity is clearly maintained (as removing an edge from a cycle does not break connectivity) and the weight of the solution increases by at least 1. This procedure is repeated until H' is either a spanning tree or a Hamiltonian cycle.

(d) Suppose $H' = (V, E \cap E_1)$ is a spanning tree. If H' is a path then the end-points of this path, say u and v, have degree 1 in H', hence we can add the edge $\{u, v\}$ and obtain a solution of greater weight. So let us suppose that H' is not a path, hence

there exists a vertex v of degree at least 3 in H'. This implies that there exists an edge $e = \{v, v_i\}$ such that $e \notin E_1$. Let $\{u, v\}$ be an edge incident to v in the spanning tree H'. Consider the spanning forest $H' - \{\{u, v\}\}$, consisting of two sub-tress H'_u and H'_v containing u and v respectively. Select a leaf $w_1 \in H'_u$ and a leaf $w_2 \in H'_v$ ($w_2 \neq v$), and set $E_1 \leftarrow E_1 - \{\{u, v\}\} \cup \{\{v, v_i\}, \{w_1, w_2\}\}$. Clearly the resulting graph is connected and has greater weight.

The above transformation rules can be applied in polynomial time to obtain a graph G_3 that is a solution of MDBCS$_d$ in G' and satisfies the conditions described in the statement of the claim. □

Suppose that there exists a PTAS for MDBCS$_d$ realized by an approximation scheme $\mathcal{A}_\delta$. This family of algorithms takes as input a graph G'' and a parameter $\delta > 0$, and returns a solution of MDBCS$_d$ of weight at least $(1 - \delta)OPT_{G''}$, where $OPT_{G''}$ is the value of an optimum solution of MDBCS$_d$ in G''.

Using this scheme, we now proceed to construct a PTAS for TSP$(1, 2)$. Given a graph G, an instance of TSP$(1, 2)$, and $\varepsilon > 0$, do the following.

1. Apply the transformation described before Claim 5.1 to G and obtain the graph G'.
2. Fix $\delta = h(\varepsilon, d)$ (to be specified later) and run $\mathcal{A}_\delta$ on G'. Let G'' be the resulting solution.
3. Apply the polynomial time transformation described in Claim 5.1 on G'', the solution obtained by $\mathcal{A}_\delta$ on G'. Let the new solution be $G^* = (V \cup V_1, E^*)$.
4. Return $E^* \cap E$ as the solution of TSP$(1, 2)$.

Now we prove that the solution returned by our algorithm satisfies $\sum_{e \in E^* \cap E} f(e) \leq (1+\varepsilon)O_T$, where O_T is the weight of an optimum tour in G. Let such an optimum tour contain a edges of weight 1 and b edges of weight 2. Then $O_T = a + 2b$ and $a + b = n$. Equivalently $a = 2n - O_T$ and $b = O_T - n$. Let O_D be the value of an optimum solution of MDBCS$_d$ in G'. Then by Claim 5.1 and the flipping nature of the function g, we have that

$$O_D = (d - 2)3n + 2a + b. \tag{5.1}$$

Let $3(d - 2)n + O^*_D$ be the value of the solution returned by $\mathcal{A}_\delta$, where O^*_D is the sum of the edge weights of the Hamiltonian cycle in G, that is, $O^*_D = \sum_{e \in E^* \cap E} g(e)$. Since $\mathcal{A}_\delta$ is a PTAS,

$$3(d - 2)n + O^*_D \geq (1 - \delta)O_D. \tag{5.2}$$

Combining Equation (5.1) and Inequality (5.2) gives

$$O^*_D \geq (1 - \delta)O_D - 3(d - 2)n = 3n - O_T + \delta O_T - n(3d - 3)\delta. \tag{5.3}$$

On the other hand, the value of the solution returned by our algorithm for TSP$(1, 2)$ is $O^*_T = 3n - O^*_D$ (since if $O^*_D = 2x + y$, x being the number of edges of weight 2 and y being the number of edges of weight 1, with $x + y = n$, then the value of the solution for TSP$(1, 2)$ is $x + 2y$). Substituting $O^*_D = 3n - O^*_T$ in Inequality (5.3) and using the fact that $O_T \geq n$ yields

$$O^*_T \leq O_T - \delta O_T + n(3d - 3)\delta \leq O_T - \delta n + n(3d - 3)\delta. \tag{5.4}$$

To show that $O_T^* \leq (1+\varepsilon)O_T$, by (5.4) it is enough to show that $-\delta n + n(3d-3)\delta \leq \varepsilon \cdot O_T$. Rather, we show that $-\delta n + n(3d-3)\delta \leq \varepsilon n$, which automatically implies the required bound. This can be done by setting $\delta = h(\varepsilon, d) = \frac{\varepsilon}{3d-4}$, yielding a PTAS for TSP$(1,2)$. Since TSP$(1,2)$ does not admit a PTAS [167], the last assertion also rules out the existence of a PTAS for MDBCS$_d$ for any $d \geq 3$, unless P = NP. □

To show the non-existence of a constant factor approximation, we first make a simplification and then define a graph product.

Remark 5.1 *We assume that the input graph $G = (V, E)$ of* MDBCS$_d$ *admits an optimal solution containing a cycle. This can be assumed without loss of generality by adding, for each vertex $v \in V$, two new vertices v_1, v_2 and a triangle consisting of the edges $\{v, v_1\}, \{v, v_2\}, \{v_1, v_2\}$ with weight ε, $0 < \varepsilon << \min_{e \in E} \omega(e)$. Let G' be the transformed graph. By the choice of ε, the new edges do not affect the structure of the optimal solutions in G. If all the optimal solutions in G were trees, these triangles could be added (for $d \geq 3$) to the leaves of any solution in G to obtain a solution in G' containing a cycle.*

We define the following *edge squaring* operation.

Definition 5.1 *Given a graph $G = (V, E)$ and an edge $\{u, v\} \in E$, define $G^2_{u,v}$ as the graph obtained from G by replacing every edge $e = \{x, y\} \in E$ with a copy G_e of G and adding the two edges $\{x, u\}$ and $\{y, v\}$ with weight ε, $0 < \varepsilon << \min_{e \in E} \omega(e)$. The vertices x and y are referred as the* contact vertices *of G_e.*

Observe that this graph product differs from the *edge square* graph introduced in [150] to prove the hardness of Longest Path, in which for every edge $e = \{x, y\} \in E$, the vertices x and y are joined to every vertex in G_e. We need this new definition for technical reasons, as the structure of the solutions of MDBCS$_d$ for $d = 2$ and for $d \geq 3$ differs considerably. The graphs $G^2_{u,v}$ are important because of the following lemma.

Lemma 5.1 *Let $G = (V, E)$ be a graph for which the total edge weight in an optimal solution for* MDBCS$_d$ *is ℓ. Then,*

(a) *there exists an edge $\{u, v\} \in E$ such that the total edge weight of an optimal solution to* MDBCS$_d$ *in $G^2_{u,v}$ is at least $\ell^2 - \ell$, and*

(b) *for any edge $\{u, v\} \in E$, given a solution of weight t in $G^2_{u,v}$, one can obtain in polynomial time a solution of weight $\sqrt{t}$ in G.*

Proof: We may assume, by subdividing edges, that all the edges of G have unit weight. Let $S = (V, E')$ be a connected subgraph of G of degree at most d having ℓ edges. By Remark 5.1, we can assume that S contains a cycle C. Let $\{u, v\}$ be an edge in C. Then the removal of $\{u, v\}$ does not disconnect S. Consider the subgraph H of $G^2_{u,v}$ containing ℓ copies of S, one for each edge in S, plus the corresponding contact vertices to connect the copies. Let H' be the subgraph obtained from H by removing the edge $\{u, v\}$ in each copy of S. This subgraph H' is connected by the choice of $\{u, v\}$, has maximum degree at most d, and has total weight $\ell^2 - \ell + 2\varepsilon\ell \geq \ell^2 - \ell$, proving (a).

Consider any solution H in $G^2_{u,v}$ with t edges which we want to use in order to find a solution S in G. We can ignore the edges of weight ε. Observe that subgraph H can pass from one copy of G corresponding to an edge to another copy of G corresponding to another edge only via contact vertices. To define S we distinguish two cases:

- **Case a:** H intersects at least $\sqrt{t}$ copies of G in $G^2_{u,v}$.
  Then let S be the subgraph of G induced by the edges corresponding to these copies of G in $G^2_{u,v}$.

- **Case b:** H intersects strictly fewer than $\sqrt{t}$ copies of G.
  Then let S_1 be $H \cap G_e$, with G_e being the copy of G in $G^2_{u,v}$ such that $|E(H \cap G_e)|$ is maximized. Let $e = \{x, y\}$. If S_1 is connected, we set $S = S_1$. Otherwise, S_1 has exactly two connected components C_1 and C_2, containing u and v, respectively. Since H is connected and S_1 is disconnected, necessarily the edges $\{x, u\}$ and $\{y, v\}$ belong to H. Therefore, as $\Delta_H \leq d$, the vertices u and v have degree at most $d-1$ in S_1. Let S be the graph obtained from S_1 by adding the edge $\{u, v\}$. (Recall that $G^2_{u,v}$ is defined only for $\{u, v\}$ being an edge of G.)

  In both cases S is connected, has maximum degree at most d, and has at least $\sqrt{t}$ edges. Since at least one of the cases must occur, it follows that given a solution of size t in $G^2_{u,v}$, one can obtain in polynomial time a solution of size $\sqrt{t}$ in G.

Part (b) of the lemma follows. □

Using Lemma 5.1 one can show the following lemma, similar to [150, Theorem 8].

Theorem 5.2 *For any $d \geq 2$, if* MDBCS$_d$ *has a polynomial time algorithm that achieves a constant factor approximation, then it has a* PTAS.

Proof: Again, we prove the result for $d \geq 3$, as the result for $d = 2$ follows from [150]. Let $\mathcal{A}$ be an algorithm that achieves a C-approximation for MDBCS$_d$, for a constant $C > 1$. Given the input graph $G = (V, E)$, we build the graphs $G^2_{u,v}$ for each edge $\{u, v\} \in E$, and inductively iterate this construction p times, where p is an integer to be specified later. Denote by $\mathcal{G}^{2^k}$ the set of all graphs obtained from G after k iterations.

Let OPT be the weight of an optimal solution to MDBCS$_d$ in G, and let OPT_{2^k} be the maximum weight over all graphs H in $\mathcal{G}^{2^k}$ of an optimal solution to MDBCS$_d$ in H. By Lemma 5.1(a),

$$OPT_{2^p} \geq OPT^2_{2^{p-1}} - OPT_{2^{p-1}} = OPT^2_{2^{p-1}} - o(OPT^2_{2^{p-1}}) \geq \ldots \geq OPT^{2^p} - o(OPT^{2^p}). \quad (5.5)$$

The PTAS is now obtained as follows. We run algorithm $\mathcal{A}$ on each graph $H \in \mathcal{G}^{2^p}$, and then pick the best solution among them. This yields a weight at least OPT_{2^p}/C, since $\mathcal{A}$ is a C-approximation algorithm. Beginning from this solution, we apply Lemma 5.1(b) p times to obtain a solution to MDBCS$_d$ in G with weight SOL, such that

$$SOL \geq \left(\frac{OPT_{2^p}}{C}\right)^{1/2^p} \geq \frac{\left(OPT^{2^p} - o(OPT^{2^p})\right)^{1/2^p}}{C^{1/2^p}} = \frac{OPT}{C^{1/2^p}} - o(OPT), \quad (5.6)$$

where we have used Equation (5.5). It is then clear from Equation (5.6) that for any $\varepsilon > 0$, there exists an integer $p(\varepsilon)$ such that for any graph G with $n = |V(G)|$ large enough, $\frac{OPT}{SOL} \leq 1 + \varepsilon$. Since for any fixed $\varepsilon > 0$, the number of graphs in $\mathcal{G}^{2^{p(\varepsilon)}}$ is polynomial in n, we have constructed a polynomial time $(1 + \varepsilon)$-approximation algorithm for any $\varepsilon > 0$. In other words, MDBCS$_d$ admits a PTAS for any $d \geq 3$, as claimed. □

Theorems 5.1 and 5.2 together yield the following.

Theorem 5.3 *The* MDBCS$_d$ *problem, for* $d \geq 2$*, does not admit any constant-factor approximation, unless* P = NP*.*

Karger *et al.* [150] also rule out an existence of weaker approximation for finding a longest path in a given graph. In the same spirit we show the following theorem.

Theorem 5.4 *If there is a polynomial time algorithm for* MDBCS$_d$*,* $d \geq 2$*, with performance ratio* $2^{O(\sqrt{\log n})}$*, then* NP $\subseteq$ DTIME$(2^{O(\log^5 n)})$*.*

Proof: Let $\mathcal{A}$ be an algorithm of approximation ratio $2^{O(\sqrt{\log n})}$ for MDBCS$_d$. Let $G = (V, E)$ be an instance to MDBCS$_d$ with n vertices and having an optimum solution containing ℓ edges. We choose p to be the smallest integer such that $N = n^{3^p} \geq 2^{\log^5 n}$. Let $g(n) = 2^{O(\sqrt{\log n})}$.

Now we generate the set of graphs $\mathcal{G}^{2^p}$ (defined in the proof of Theorem 5.2) by applying p times the edge squaring operation starting with G and then repeatedly applying it on previously obtained graphs. Note that the number of vertices of the elements in $\mathcal{G}^{2^p}$ is bounded above by N. By Lemma 5.1(a) we know that $\mathcal{G}^{2^p}$ contains a graph which has a solution of size at least $\ell^{2^p} - o(\ell^{2^p})$. Running $\mathcal{A}$ on all the elements of $\mathcal{G}^{2^p}$ and picking the graph with the largest number of edges, we obtain a solution H of size at least $\left(\ell^{2^p} - o(\ell^{2^p})\right)/g(N)$. Since $p = O(\log \log n)$, the number of graphs in $\mathcal{G}^{2^p}$ is bounded above by $n^{2^p} = n^{O(\log n)} = 2^{O(\log^2 n)}$. Furthermore, starting with the solution H and repeatedly applying Lemma 5.1(b), we obtain a solution to MDBCS$_d$ in G of size at least $(\ell - o(\ell))/h(n)$, where

$$h(n) = g(N)^{1/2^p} = 2^{O(\sqrt{\log N}/2^p)} = 2^{O(\log^{2.5} n/2^p)} = O(1) .$$

The last equality follows because $2^p \geq \log^{2.5} n$. Since $p = O(\log \log n)$ it follows that we have a constant factor approximation algorithm for $\ell = \Omega(\log n/\log \log n)$. If $\ell \leq c \log n/\log \log n$, for a fixed constant c, then the problem can be solved exactly in time $n^{O(\log n/\log \log n)} = 2^{O((\log n)^2/\log \log n)} \leq 2^{O(\log^5 n)}$. To find an exact solution, proceed as follows. Enumerate all subgraphs on $2c \log n/\log \log n$ vertices and at most $c \log n/\log \log n$ edges. Notice that the number of such subgraphs is upper bounded by $2^{O(\log n \log \log n)}$. Let H be one of the enumerated graphs. Then we check whether its maximum degree is upper bounded by d and whether it is connected. If both requirements are met then we check whether there is a subgraph isomorphic to H in G. To do so we use the result of Alon *et al.* [34], which shows that if H has treewidth t and $|V(H)| = h$ then one can check whether there is a subgraph isomorphic to H in G in $2^{O(h)} n^{t+1} \log n$ time. Observe that the treewidth of H is upper bounded by $\ell \leq c \log n/\log \log n$. Hence we can find an optimum solution in time

$$2^{O(\log n \log \log n)} \cdot 2^{O(\log n/\log \log n)} n^{(\log n/\log \log n)+1} \log n \leq 2^{O(\log^2 n)}.$$

This implies that we can approximate MDBCS$_d$ within a constant factor in time $2^{O(\log^5 n)}$, which is polynomial in N. But by Theorem 5.3 we know that finding a constant factor approximation to MDBCS$_d$ is NP-hard, hence we have given a simulation of a NP-hard problem in time $2^{O(\log^5 n)}$. The theorem follows. □

5.3 Approximating MDBCS$_d$

In this section we focus on approximating MDBCS$_d$. As seen in Section 5.2, MDBCS$_d$ does not admit any constant-factor approximation in general graphs.

First, we provide in Section 5.3.1 approximation algorithms in general graphs for both the weighted and unweighted versions of the problem. Then, we show in Section 5.3.2 that when the input graph has a low-degree spanning tree (in terms of d), the problem becomes easy to approximate in unweighted graphs. Specifically, Proposition 5.2 provides a constant-factor approximation for such graphs.

5.3.1 General graphs

In this section we deal with general graphs. Concerning the Longest Path problem (which corresponds to the case $d = 2$ of MDBCS$_d$ as discussed in the introduction) the best currently known approximation algorithm [56] has approximation ratio $O\left(n(\frac{\log\log n}{\log n})^2\right)$ (which was an improvement on the approximation ratio $O(n/\log n)$ of the algorithm in [34]). Using the results of [34], we provide in Theorem 5.6 an approximation algorithm for MDBCS$_d$ in general unweighted graphs for any $d \geq 2$. We then turn to weighted graphs and provide a new approximation algorithm for general weighted graphs in Theorem 5.7. Finally we compare both algorithms for unweighted graphs.

We need a preliminary lemma, that uses the following result.

Proposition 5.1 ([165]) *Any unordered tree on n nodes can be represented using $2n+o(n)$ bits with adjacency being supported in $O(n)$ time.*

Let $\mathcal{T}_{n,d}$ be the set of non-isomorphic unlabeled trees on n nodes with maximum degree at most d.

Lemma 5.2 *The set $\mathcal{T}_{\log n,d}$ can be generated in polynomial time on n.*

Proof: It is well known that $|\mathcal{T}_{n,n-1}| \sim C\alpha^n n^{-5/2}$ as $n \to \infty$, for positive constants C and α, c.f. [172]. Hence, the set $\mathcal{T}_{\log n,\log n-1}$ is of size polynomial in n. In addition, one can efficiently generate all the elements of $\mathcal{T}_{\log n,\log n-1}$. Indeed by Proposition 5.1 any unlabeled tree on $\log n$ nodes can be represented using $2\log n + o(\log n)$ bits with adjacency being supported in $O(\log n)$ time. Finally, the set $\mathcal{T}_{\log n,d}$ is obtained from $\mathcal{T}_{\log n,\log n-1}$ by removing all the elements T with $\Delta_T > d$, where Δ_T is the maximum degree of the tree T. □

The main ingredient of the first algorithm is a powerful result of [34], which uses the *color-coding* method.

Theorem 5.5 ([34]) *If a graph $G = (V, E)$ contains a subgraph isomorphic to a graph $H = (V_H, E_H)$ whose treewidth is at most t, then such a subgraph can be found in $2^{O(|V_H|)} \cdot |V|^{t+1} \cdot \log|V|$ time.*

In particular, trees on $\log|V|$ vertices can be found in time $|V|^{O(1)} \cdot \log|V|$. We are ready to describe our algorithm for unweighted graphs.

Algorithm $\mathcal{A}$:

(1) Generate all the elements of $\mathcal{T}_{\log n,d}$. Define the set $\mathcal{F} \leftarrow \emptyset$.

(2) For each $T \in \mathcal{T}_{\log n,d}$, test if G contains a subgraph isomorphic to T. If such a subgraph is found, add it to $\mathcal{F}$.

(3) If $\mathcal{F} = \emptyset$ OR $d > \log n$, output an arbitrary connected subgraph of G with d edges. Otherwise, output any element in $\mathcal{F}$.

Theorem 5.6 *For all $d \geq 2$, algorithm $\mathcal{A}$ provides a ρ-approximation algorithm for* MDBCS$_d$ *in unweighted graphs, with $\rho = \min\{m, nd/2\}/\log n$.*

Proof: Let us first observe that the running time of algorithm $\mathcal{A}$ is polynomial in n. Indeed, steps **(1)** and **(2)** can be executed in polynomial time by Lemma 5.2 and Theorem 5.5, respectively. Step **(3)** takes constant time. Algorithm $\mathcal{A}$ is clearly correct, since by definition of the set $\mathcal{T}_{\log n,d}$ the output graph is a solution of MDBCS$_d$ in G.

Finally, let us consider the approximation ratio of algorithm $\mathcal{A}$. Let OPT be the number of edges of an optimal solution of MDBCS$_d$ in G, and let ALG be the number of edges of the solution found by algorithm $\mathcal{A}$. We distinguish two cases:

- If $OPT \geq \frac{d \cdot \log n}{2}$, then any optimal solution $\hat{H}$ has at least $\log n$ vertices. In particular, $\hat{H}$ contains a tree on $\log n$ vertices, and so does G. Hence, this tree will be found in step **(2)**, and therefore $ALG \geq \log n - 1$. (We can assume that $ALG = \log n$ by replacing everywhere $\mathcal{T}_{\log n,d}$ with $\mathcal{T}_{\log n+1,d}$.) On the other hand, we know that $OPT \leq \min\{m, nd/2\}$.
- Otherwise, if $OPT < \frac{d \cdot \log n}{2}$, then $ALG \geq d$. Note that such a connected subgraph with d edges can be greedily found starting from any node of G.

In both cases,

$$\frac{OPT}{ALG} \leq \max\left\{\frac{\min\left\{m, \frac{nd}{2}\right\}}{\log n}, \frac{\log n}{2}\right\} = \frac{\min\{m, nd/2\}}{\log n}$$

(since $\log n = O(\sqrt{n})$), as claimed. □

In particular, if $d = 2$, Algorithm $\mathcal{A}$ reduces to the LONGEST PATH algorithm of [34].

Theorem 5.7 *The* MDBCS$_d$ *problem admits a ρ-approximation algorithm $\mathcal{B}$ in weighted graphs, with $\rho = \min\{n/2, m/d\}$.*

Proof: Let us describe the algorithm $\mathcal{B}$. Let F be the set of d heaviest edges in the input graph G, and let W be the set of endpoints of those edges. We distinguish two cases according to the connectivity of the subgraph $H = (W, F)$. Let $\omega(F)$ denote the total weight of the edges in F.

If H is connected, the algorithm returns H. We claim that this yields a ρ-approximation. Indeed, if an optimal solution consists of m^* edges of total weight ω^*, then $ALG = \omega(F) \geq \frac{\omega^*}{m^*} \cdot d$, since by the choice of F the average weight of the edges in F cannot be smaller than the average weight of the edges of an optimal solution. As $m^* \leq m$ and $m^* \leq dn/2$, we get that $ALG \geq \frac{\omega^*}{m} \cdot d$ and $ALG \geq \frac{\omega^*}{dn/2} \cdot d = \frac{\omega^*}{n/2}$.

Now suppose $H = (W, F)$ consists of a collection $\mathcal{F}$ of k connected components. Then we *glue* these components together in $k-1$ phases. In each phase, we pick two components $C, C' \in \mathcal{F}$, and combine them into a new connected component $\hat{C}$ by adding a connecting path, without touching any other connected component of $\mathcal{F}$. We then set $\mathcal{F} \leftarrow \mathcal{F} \setminus \{C, C'\} \cup \{\hat{C}\}$.

Each phase operates as follows. For every two components $C, C' \in \mathcal{F}$, compute their distance, defined as $d(C, C') = \min\{dist(u, u', G) \mid u \in C, u' \in C'\}$. Take a pair $C, C' \in \mathcal{F}$ attaining the smallest distance $d(C, C')$. Let $u \in C$ and $u' \in C'$ be two vertices realizing this distance, i.e., such that $dist(u, u', G) = d(C, C')$. Let $p(u, u')$ be a shortest path between u and u' in G. Let $\hat{C}$ be the connected component obtained by merging C, C' and the path $p(u, u')$.

For the correctness proof, we need the following two observations: First, observe that in every phase, the path $p(u, u')$ used to merge the components C and C' does not go through any other cluster C'', since otherwise, $d(C, C'')$ would be strictly smaller than $d(C, C')$, contradicting the choice of the pair (C, C'). Moreover, $p(u, u')$ does not go through any other vertex v in the cluster C except for its endpoint u, since otherwise, $dist(v, u', G) < dist(u, u', G)$, contradicting the choice of the pair u, u'. Similarly, $p(u, u')$ does not go through any other vertex v' in C'.

We now claim that after i phases, the maximum degree of H satisfies $\Delta_H \leq d - k + i + 1$. This is proved by induction on i. For $i = 0$, i.e., for the initial graph $H = (W, F)$, we observe that as F consists of d edges arranged in k separate components, the largest component will have no more than $d - k + 1$ edges, hence $\Delta_H \leq d - k + 1$, as required. Now suppose the claim holds after $i - 1$ phases, and consider phase i. All nodes other than those of the path $p(u, u')$ maintain their degree from the previous phase. The nodes u and u' increase their degree by 1, so by the inductive hypothesis, their new degree is at most $(d - k + (i - 1) + 1) + 1 = d - k + i + 1$, as required. Finally, the intermediate nodes of $p(u, u')$ have degree $2 \leq d - k + i + 1$ (since $i \geq 1$ and $k \leq d$).

It follows that by the end of phase $k - 1$, $\Delta_H \leq d - k + k - 1 + 1 = d$. Also, at that point H is connected. Hence H is a valid solution.

Finally, the approximation ratio of the algorithm is still at most $\rho = \min\{n/2, m/d\}$, since this ratio was guaranteed for the originally selected F, and the final subgraph contains the set F. □

For unweighted graphs, comparing approximation ratios of Algorithm $\mathcal{A}$ of Theorem 5.6 and Algorithm $\mathcal{B}$ of Theorem 5.7, we conclude that Algorithm $\mathcal{A}$ performs better when $d < 2 \log n$, while Algorithm $\mathcal{B}$ is better when $d \geq 2 \log n$. So if we run both the algorithms and select the best solution, we obtain the following.

Corollary 5.1 *In unweighted graphs, the* MDBCS$_d$ *problem admits a ρ-approximation algorithm, with $\rho = \min\{n/2,\ nd/(2 \log n),\ m/d,\ m/\log n\}$.*

5.3.2 Graphs with low-degree spanning trees

We first state a simple lemma about the optimal solutions of the (polynomially solvable) problem MDBS_d (the definition is the same as the MDBCS_d problem, except that the connectivity of the output subgraph is not required).

Lemma 5.3 *Given a graph G and two integers d, k , $1 < k \leq d$, such that k divides d, let OPT_d and $OPT_{d/k}$ be the optimal solutions of* MDBS_d *and* $\text{MDBS}_{d/k}$ *in G, respectively. Then $OPT_d \leq \frac{3k}{2} \cdot OPT_{d/k}$.*

Proof: Let $\hat{H}_d$ be the subgraph of G attaining OPT_d. By the classical Vizing's theorem [95], there exists a coloring of the edges of $\hat{H}_d$ using at most $d+1$ colors. Order these chromatic classes according to non-increasing total edge-weight, and let $H_{d/k}$ be the subgraph of G induced by the first d/k classes. Then the maximum degree of $H_{d/k}$ does not exceed d/k, and the sum of its edge weights is at least $\frac{d \cdot OPT_d}{k \cdot (d+1)}$. Hence

$$OPT_d \leq \frac{d+1}{d} \cdot k \cdot OPT_{d/k} \ .$$

For $d \geq 2$, the function $\frac{d+1}{d}$ is maximized when $d = 2$. □

For example, if $G = C_5$ and $d = k = 2$, then $OPT_2 = 5 \leq 3/2 \cdot 2 \cdot OPT_1 = 3 \cdot 2 = 6$.

Definition 5.2 (k-tree) *A* k-tree *of a connected graph is a spanning tree with maximum degree at most k.*

We are now ready to describe our approximation algorithm.

Proposition 5.2 *Given two integers d, ℓ , $1 < \ell < d$, let $\mathcal{G}_{d,\ell}$ be the class of graphs that have a $(d/\ell - 1)$-tree. Then, for any $G \in \mathcal{G}_{d,\ell}$,* MDBCS_d *can be approximated in G within a constant factor $\frac{3}{2}\frac{\ell}{\ell-1}$.*

Proof: Assume without loss of generality that ℓ divides d (otherwise, replace d/ℓ with $\lceil d/\ell \rceil$). Since G has a $(d/\ell - 1)$-tree, by [128], one can find in polynomial time a spanning tree S of G with maximum degree at most d/ℓ. Let $k = \frac{\ell}{\ell-1}$, and let H be the optimal solution of $\text{MDBS}_{d/k}$ in G (recall that MDBS_d is in P, but the output graph is not necessarily connected). Then it is clear that the graph $S \cup H$ is a solution of MDBCS_d in G, since it is connected and has maximum degree at most d. By Lemma 5.3 and using the fact that any solution for MDBCS_d is also a solution for MDBS_d, we conclude that $S \cup H$ provides a $\frac{3}{2}\frac{\ell}{\ell-1}$-approximation for MDBCS_d in G. □

For example, if G has a spanning tree of maximum degree at most $d/2 - 1$, then Proposition 5.2 states that MDBCS_d admits a 3-approximation in G.

The relation between MDBCS$_d$ and graph toughness

Given a graph G, denote by $\kappa(G)$ the number of connected components of G.

Definition 5.3 (Toughness of a graph [195]) *The* toughness $t(G)$ *of a graph* $G = (V, E)$ *is the largest number* t *such that, for any subset* $S \subseteq V$, $|S| \geq t \cdot \kappa(G[V \setminus S])$, *provided that* $\kappa(G[V \setminus S]) > 1$.

It is proved in [195] that if $t(G) \geq \frac{1}{k-2}$, for $k \geq 3$, then G has a k-tree.

Theorem 5.8 ([195]) *Let* G *be a graph. If* $t(G) \geq \frac{1}{k-2}$, *with* $k \geq 3$, *then* G *has a* k*-tree.*

Let us relate the above definitions with the MDBCS$_d$ problem. If a graph G does not satisfy the conditions of Proposition 5.2, then G does not have a $(d/2-1)$-tree. In this case one has some additional knowledge about the structure of G. Namely, Theorem 5.8 states that, provided that $d \geq 8$, the *toughness* $t(G)$ of G satisfies $t(G) < \frac{1}{d/2-3}$, implying that there exists a subset $S \subseteq V(G)$ such that

$$\kappa(G[V \setminus S]) > |S| \cdot \left(\frac{d}{2} - 3\right).$$

It would be interesting to explore the question whether this structural result permits to approximate MDBCS$_d$ in G efficiently.

5.4 Hardness of Approximating MSMD$_d$

The main result of this section, Theorem 5.12, states that MSMD$_d$ does not admit a constant-factor approximation on general graphs, for $d \geq 3$. We first prove in Section 5.4.1 that MSMD$_d$ does not admit a PTAS, and then use the error amplification technique to prove the main result. Our reduction is obtained from the well known Vertex Cover (VC) problem (see Section I.2.1).

5.4.1 MSMD$_d$ does not admit a PTAS for any $d \geq 3$

The result is first established for the case $d = 3$, in Theorem 5.9, and then extended to any $d \geq 3$ in Theorem 5.10.

Theorem 5.9 *The* MSMD$_3$ *problem does not admit a* PTAS, *unless* P = NP.

Proof: We present a gap-preserving reduction from Vertex Cover. Let H be an instance of Vertex Cover on n vertices. We construct an instance $G = f(H)$ of MSMD$_3$. Without loss of generality, we may assume that H contains $3 \cdot 2^m$ edges for some integer m, and also that every vertex of H has degree at least three.

Let T be the complete ternary rooted tree with root r and height $m+1$. T contains $3 \cdot 2^m$ leaves and $3 \cdot 2^{m+1} - 2$ vertices overall. Let us identify the leaves of T with edges of H,

and call this set E (note that $E \subseteq V(T)$). We add another copy of E, called F, and a Hamiltonian cycle on $E \cup F$ inducing a bipartite graph with partition classes E and F, as shown in Figure 5.1. Let us also identify the vertices of F with edges in H. Now we add n new vertices A identified with vertices of H, and join them to the leaves of T according to the adjacency relations between the edges and vertices in H, i.e., an element $\ell \in T$ is connected to $v \in A$ if the edge corresponding to ℓ in H is adjacent to the vertex v of $V(H)$. The graph G built in this way is depicted in Figure 5.1.

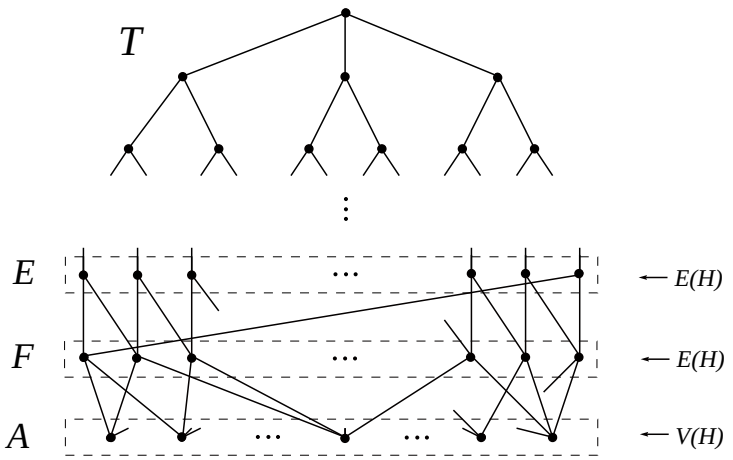

Figure 5.1: An example of the graph G built in the reduction of Theorem 5.9.

We now claim that minimum subgraphs of G of minimum degree at least three correspond to minimum vertex covers of H and vice versa. To see this, first note that if such a subgraph U of G contains a vertex of $T \cup F$, then it should contain all the vertices of $T \cup F$, because of the degree constraints. Obviously U cannot consist just of vertices of A, hence U must contain all the vertices of $T \cup F$. Note that all the vertices of F have degree two in $G[T \cup F]$. Therefore, the problem reduces to finding the smallest subset of vertices in A covering all the vertices in F. This is exactly the Vertex Cover problem for H. Thus, we have that

$$OPT_{\mathrm{MSMD}_3}(G) \;=\; OPT_{\mathrm{VC}}(H) + |V(T)| + |V(F)| \;=\; OPT_{\mathrm{VC}}(H) + 9 \cdot 2^m - 2 \;.$$

Using this formula, it is straightforward to check that f is a gap-preserving reduction (see Section I.2.1). To complete the proof, note that Vertex Cover is Apx-hard, even restricted to graphs H of size linear in $OPT_{\mathrm{VC}}(H)$. The existence of a PTAS for MSMD_3 provides a PTAS for Vertex Cover, leading to contradiction (under the assumption that Apx $\neq$ PTAS). □

Theorem 5.10 *The* MSMD_d *problem, for* $d \geq 3$*, does not admit a* PTAS*, unless* P = NP*.*

Proof: The case $d = 3$ has been proved in Theorem 5.9, so assume $d \geq 4$. Again, the proof is based on a *gap-preserving reduction* from Vertex Cover. Let H be an instance

of Vertex Cover on n vertices. We construct an instance $G = f(H)$ of MSMD$_d$. Without loss of generality, we may suppose that H contains $d \cdot (d-1)^m$ edges, for some m, and also that every vertex of H has degree at least d.

Let T be the complete d-ary rooted tree with root r and height $m+1$. It is easy to see that T contains $d \cdot (d-1)^m$ leaves and $1 + d \cdot \frac{(d-1)^{m+1}-1}{d-2}$ vertices in total. Let us identify the leaves of T with edges of H, and call this set E (note that $E \subseteq V(T)$). We add another copy of E, called F, and the following edges (assuming that m is big enough) according to the parity of d:

- if d is even: $\frac{d-2}{2}$ Hamiltonian cycles on $E \cup F$, each inducing a bipartite graph with partition classes E and F, plus one perfect matching between E and F.
- if d is odd: $\frac{d-1}{2}$ Hamiltonian cycles on $E \cup F$, each inducing a bipartite graph with partition classes E and F.

Let us also identify the vertices of F with edges in H. Now we add n new vertices A identified with vertices of H, and join them to the leaves of T according to the adjacency relations between the edges and vertices in H, i.e., an element $\ell \in T$ is connected to $v \in A$ if the edge corresponding to ℓ in H is adjacent to the vertex v of $V(H)$. Note that the vertices of E have regular degree d, and those of F have regular degree $d+1$. Now the same idea of the proof of Theorem 5.9 applies to this case, proving the Apx-hardness of MSMD$_d$ for $d > 3$. □

5.4.2 MSMD$_d$ is not in Apx for any $d \geq 3$

We are now ready to prove the main result of this section. Again, we focus on the case $d = 3$ in Theorem 5.11 and then extend the ideas for any $d \geq 3$ in Theorem 5.12.

Theorem 5.11 *The* MSMD$_3$ *problem does not admit any constant-factor approximation, unless* P = NP.

Proof: The proof is by appropriately applying the standard error amplification technique. Let $\mathcal{G}_1 = \{G\}$ be the family of graphs constructed in Theorem 5.9 (see Figure 5.1) from the instances H of vertex cover, G being a typical member of this family, and let $\alpha > 1$ be the factor of inapproximability of MSMD$_3$, that exists by Theorem 5.9.

We construct a sequence of families of graphs $\mathcal{G}_k$, such that MSMD$_3$ is hard to approximate within a factor $\theta(\alpha^k)$ in the family $\mathcal{G}_k$. This proves that MSMD$_3$ does not have any constant-factor approximation. In the following, G_k will denote a typical element of $\mathcal{G}_k$ constructed using the element G of $\mathcal{G}_1$. We describe the construction of G_2, and obtain the result by repeating the same construction inductively to obtain G_k. For every vertex v in G, we construct a graph G_v as follows. First, letting $d_v = deg_G(v)$, take a copy of G and choose d_v other arbitrary vertices $x_1, \ldots, x_{d_v}$ of degree three in $T \subset G$. Then, replace each of these vertices x_i with a cycle of length four, and join three of the vertices of the cycle to the three neighbors of x_i, $i = 1, \ldots, d_v$. Let G_v be the graph obtained in this way. Note that it contains exactly d_v vertices of degree two in G_v.

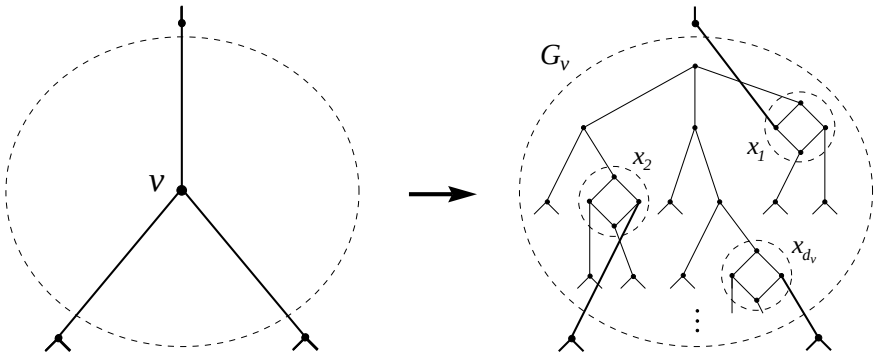

Figure 5.2: Error amplification in the proof of Theorem 5.11.

Now take a copy of G, and replace each vertex v with G_v. Then, join the d_v edges incident to v to the d_v vertices of degree two in G_v. This completes the construction of the graph G_2, which is illustrated in Figure 5.2.

We have that $|V(G_2)| = |V(G)|^2 + o(|V(G)|^2)$, because each vertex of G is replaced with a copy of G where we had replaced some of the vertices with a cycle of length four.

To find a solution of MSMD_3 in G_2, note that for any $v \in V(G)$, once a vertex in G_v is chosen, we have to look for MSMD_3 in G, which is hard up to a constant factor α. But approximating the number of v's for which we should touch G_v is also MSMD_3 in G, which is hard up to the same factor α. This proves that approximating MSMD_3 in G_2 is hard up to a factor α^2. The proof of the theorem is completed by repeating this procedure, applying the same construction to obtain G_3, and inductively G_k. □

Theorem 5.12 *The* MSMD_d *problem, for* $d \geq 3$*, does not admit any constant-factor approximation, unless* P = NP*.*

Proof: Again, the proof is based on applying the error amplification technique. Let $G_1 = G$ be the graph constructed in Theorem 5.10, and let $\alpha > 1$ be the factor of inapproximability of MSMD_d, that exists by Theorem 5.10. We construct a sequence of graphs $\mathcal{G}_k$, such that MSMD_d is hard to approximate within a factor $\theta(\alpha^k)$ in $\mathcal{G}_k$. This proves that MSMD_d does not have any constant-factor approximation. Indeed, suppose that MSMD_d admits a C-approximation for some constant $C > 0$. Then we can choose k such that $\alpha^k > C$, and then MSMD_d is hard to approximate in $\mathcal{G}_k$ within a factor $\alpha^k > C$, a contradiction.

We describe the construction of G_2, and obtain the result by repeating the same construction inductively to create G_k, a typical element of $\mathcal{G}_k$. For every vertex v in G, construct a graph G_v as follows. First, take a copy of G, and choose $d_v = deg_G(v)$ other arbitrary vertices $x_1, \ldots, x_{d_v}$ of degree d in $T \subset G$. Then, replace each of these vertices x_i with the following:

- if d is odd: a graph on $d + 1$ vertices with regular degree $d - 1$.
- if d is even: a graph on $d + 2$ vertices having one vertex v^* of degree $d + 1$, and all the others of degree $d - 1$.

Next, join d of the vertices of this new graph (different from v^*) to the d neighbors of x_i, $i = 1, \ldots, d_v$. Let G_v be the graph obtained in this way. Note that we have exactly d_v vertices of degree $d-1$ in G_v.

Now, take a copy of G, and replace each vertex v with G_v. Then, join the d_v edges incident to v to the d_v vertices of degree $d-1$ in G_v. This completes the construction of the graph G_2.

We have that $|V(G_2)| = |V(G)|^2 + o(|V(G)|^2)$, because each vertex of G is replaced with a copy of G where we had replaced some of the vertices with a graph of size $d+1$ or $d+2$. The same idea of the proof of Theorem 5.11 applies to this case, proving the APX-hardness of MSMD_d for $d > 3$. □

5.5 Approximating MSMD_d

In this section, it is shown that for fixed d, MSMD_d is in P for graphs whose treewidth is $O(\log n)$. This is done by presenting a polynomial time algorithm based on dynamic programming. We refer to Section Section I.1.1 (page 15) for the definitions of tree decomposition and treewidth.

This dynamic programming algorithm is then used in Section 5.5.2 to provide an $O(\frac{n}{\log n})$-approximation algorithm of MSMD_d for all classes of graphs excluding a fixed graph as a minor. This algorithm relies on a partitioning result for minor-excluded class of graphs, proved by Demaine et al. in [90].

5.5.1 MSMD_d is in P for graphs with small treewidth

In order to prove our results, we need the following lemma which gives the time complexity of finding a smallest induced subgraph of degree at least d in graphs of bounded treewidth.

Lemma 5.4 (Lemma 6.1 of Chapter 6) *Let G be a graph on n vertices with a tree decomposition of width at most t, and let d be a positive integer. Then in time $O((d+1)^t(t+1)^{d^2}n)$, one can either find a smallest induced subgraph of minimum degree at least d in G, or identify that no such subgraph exists.*

A graph G is *q-degenerated* if every induced subgraph of G has a vertex of degree at most q. It is well known that there is a constant c such that for every h, every graph with no K_h minor is $ch\sqrt{\log h}$-degenerated [95]. This implies that M-minor-free graphs with $|M| = h$ are $ch\sqrt{\log h}$-degenerated and hence the largest value of d for which MSMD_d is non-empty is $ch\sqrt{\log h}$, a constant. The above discussion, combined with the time complexity analysis mentioned in Lemma 5.4, imply the following.

Corollary 5.2 *Let G be an n-vertex graph excluding a fixed graph M as minor, with a tree decomposition of width $O(\log n)$, and let d be a positive integer constant. Then in polynomial time one can either find a smallest induced subgraph of minimum degree at least d in G, or conclude that no such subgraph exists.*

5.5.2 Approximation algorithm for M-minor-free graphs

The following result of Demaine *et al.* [90] provides a way for partitioning the vertices of a graph excluding a fixed graph as a minor into subsets with small treewidth.

Theorem 5.13 ([90]) *For a fixed graph M, there is a constant c_M such that for every integer $k \geq 1$ and for every M-minor-free graph G, the vertices of G (or the edges of G) can be partitioned into $k+1$ sets such that any k of the sets induce a graph of treewidth at most $c_M k$. Furthermore, such a partition can be found in polynomial time.*

One may assume without loss of generality that the minimum degree of the minor-free input graph $G = (V, E)$ is at least d (by removing all the vertices of lower degree), and also that $|V(G)| = n = 2^p$ for some integer $p \geq 0$ (otherwise, replace $\log n$ with $\lceil \log n \rceil$ in the description of the algorithm).

Description of the algorithm:

(1) Relying on Theorem 5.13, partition $V(G)$ in polynomial time into $\log n + 1$ sets $V_0, \ldots, V_{\log n}$ such that any $\log n$ of the sets induce a graph of treewidth at most $c_M \log n$, where c_M is a constant depending only on the excluded graph M.

(2) Run the dynamic programming algorithm of Section 5.5.1 on all the subgraphs $G_i = G[V \setminus V_i]$ of $\log n$ sets, $i = 0, \ldots, \log n$.

(3) This procedure finds all the solutions of size at most $\log n$. If no solution is found, output the whole graph G.

This algorithm clearly provides an $O(n/\log n)$-approximation for MSMD_d in minor-free graphs, for all $d \geq 3$. The running time of the algorithm is polynomial in n, since in step **(2)**, for each G_i, the dynamic programming algorithm runs in $O((d+1)^{t_i}(t_i+1)^{d^2}n)$ time, where t_i is the treewidth of G_i, which is at most $c_M \log n$.

5.6 Approximating DDDkS

We provide a deterministic approximation algorithm for the DDDkS problem in Theorem 5.14 (strongly based on the algorithm for DkS of [112]), and a randomized approximation algorithm in Theorem 5.15. Even though the performance of the randomized algorithm is worse than the one of the deterministic algorithm, we include it because the idea behind the algorithm is quite simple.

Theorem 5.14 *The* DDDkS *problem admits a deterministic $O(n^\delta)$-approximation algorithm, for some universal constant $\delta < 1/3$.*

Proof: Given an input graph G, let ρ_k^{OPT} be the optimal average degree of a subgraph of G on exactly k vertices (i.e., the optimum of DkS), and let δ_k^{OPT} be the optimal minimum degree of a subgraph of G with at most k vertices (i.e., the optimum of DDDkS). Let C be the approximation ratio of the algorithm for DkS of [112], i.e., $C = O(n^\delta)$ for some universal constant $\delta < 1/3$. Given a graph H, let ρ_H denote the average degree of H.

We know, by [112], that we can find a subgraph H_k of G on k vertices such that $\rho_{H_k} \geq \rho_k^{OPT}/C$. Removing recursively the vertices of H_k with degree strictly smaller that $\rho_{H_k}/2$ we obtain a subgraph H'_k of H_k on at most k vertices such that $\delta_{H'_k} \geq \rho_{H_k}/2 \geq \rho_k^{OPT}/(2C)$.

The next step consists of proving that there exists an integer k_0, $1 \leq k_0 \leq k$, such that $\rho_{k_0}^{OPT} \geq \delta_k^{OPT}$, so we can run the D$k$S algorithm for each $k' \leq k$, remove low-degree vertices each time, and take the best solution of DDDkS among $H'_2, H'_3, \ldots, H'_{k-1}, H'_k$.

Finally, let us prove that k_0 exists. Let H be the optimal solution of DDDkS, $\delta_H = \delta_k^{OPT}$. Let $k_0 = |V(H)|$ ($k_0 \leq k$). This is the k_0 we are looking for, because $\rho_{k_0}^{OPT} \geq \rho_H \geq \delta_H = \delta_k^{OPT}$.

The above procedure clearly constitutes a $(2C)$-approximation for DDDkS. □

Theorem 5.15 *The* DDDkS *problem admits a randomized* $O(\sqrt{n}\log n)$*-approximation algorithm.*

Proof: For every $1 \leq d \leq n$, let $H[d]$ be the maximum subgraph of G with minimum degree $\delta_{H[d]} \geq d$, in the sense that $H[d]$ contains any other subgraph H of G of minimum degree at least d. Also let $n[d] = |V(H[d])|$. The first stage of the algorithm computes $H[d]$ for every $1 \leq d \leq n$. This is easily done by initializing $H[1] = G$ and then successively removing from $H[d]$ all the vertices of degree d to obtain $H[d+1]$. Note that $n[d]$ can be zero, i.e., $H[d]$ can be the empty subgraph. The algorithm stops whenever it finds $n[d] = 0$.

Let $\tilde{d}$ be the index such that $n[\tilde{d}] > 0$ and $n[\tilde{d}+1] = 0$ (clearly $\tilde{d} \leq n-1$). If $k \geq n[\tilde{d}]$, then $H[\tilde{d}]$ is an exact solution to the problem, hence the output to the DDDkS problem is $\tilde{d}$. It remains to handle the case where $k < n[\tilde{d}]$. In this case, it is also clear that the solution d^* we are looking for is bounded by $\tilde{d}$, i.e., $d^* < \tilde{d}$. Two cases may occur.

- **Case a** : $k \leq 16\sqrt{n}\log n$ OR $\tilde{d} \leq 16\sqrt{n}\log n$.

  In this case any connected subgraph of G of size at most k (for example a connected subtree of a spanning tree of G of size k, or even just an edge) has minimum degree at least one, hence it provides a solution that is within a factor $1/(16\sqrt{n}\log n)$ of the optimal solution.

- **Case b** : Both $\tilde{d}, k > 16\sqrt{n}\log n$.

  Construct a subgraph H of $H[\tilde{d}]$ in the following way: select each vertex of $H[\tilde{d}]$ with probability $1/\sqrt{n}$, and take H to be the induced subgraph of $H[\tilde{d}]$ by the set of selected vertices. Let $n_0 = |V(H)|$.

Claim 5.2 *The number of selected vertices satisfies* $n_0 \leq 2n[\tilde{d}]/\sqrt{n}$ *with probability at least* $1 - 1/n^4$*. In particular,* $n_0 \leq k$ *with probability at least* $1 - 1/n^4$*.*

Proof: Observe that n_0 can be expressed as the sum of $n[\tilde{d}]$ independent Boolean random variables $B_1, \ldots, B_{n[\tilde{d}]}$. Since $\mathbb{E}[n_0] = n[\tilde{d}]/\sqrt{n}$, applying Chernoff's bound on the upper tail yields

$$\mathbb{P}\mathrm{rob}\left[B_1 + \cdots + B_{n[\tilde{d}]} > \frac{2n[\tilde{d}]}{\sqrt{n}}\right] < \exp\left(-\frac{n[\tilde{d}]}{4\sqrt{n}}\right).$$

Therefore, because $n[\tilde{d}] > k > 16\sqrt{n}\log n$, we have

$$\mathbb{P}\mathrm{rob}\left[n_0 > \frac{2n[\tilde{d}]}{\sqrt{n}}\right] \;<\; \exp(-4\log n) \;=\; \frac{1}{n^4}\;,$$

and since $n[\tilde{d}] \le n$, with probability at least $1 - \frac{1}{n^4}$, $n_0 \le 2n[\tilde{d}]/\sqrt{n} \le 2\sqrt{n} < 16\sqrt{n}\log n < k$. □

Claim 5.3 *For every vertex $v \in V(H)$, $deg_H(v) \ge \frac{\tilde{d}}{2\sqrt{n}}$ with probability at least $1 - 1/n^2$.*

Proof: Observe first that $deg_H(v)$ is a sum of $deg_{H[\tilde{d}]}(v)$ independent Boolean random variables, and so the expected degree of v in H is $deg_{H[\tilde{d}]}(v)/\sqrt{n} \;\ge\; \tilde{d}/\sqrt{n}$. This is because every vertex of $H[\tilde{d}]$ has degree at least $\tilde{d}$. This implies

$$\mathbb{P}\mathrm{rob}\left[deg_H(v) < \frac{\tilde{d}}{2\sqrt{n}}\right] \;\le\; \mathbb{P}\mathrm{rob}\left[deg_H(v) \;<\; \frac{deg_{H[\tilde{d}]}(v)}{2\sqrt{n}}\right].$$

Applying Chernoff's bound on the lower tail we have

$$\mathbb{P}\mathrm{rob}\left[deg_H(v) < \frac{deg_{H[\tilde{d}]}(v)}{2\sqrt{n}}\right] \;<\; \exp\left(-\frac{deg_{H[\tilde{d}]}(v)}{8\sqrt{n}}\right) \;\le\; \exp\left(-\frac{\tilde{d}}{8\sqrt{n}}\right),$$

which in turn implies (because $\tilde{d} > 16\sqrt{n}\log n$),

$$\mathbb{P}\mathrm{rob}\left[deg_H(v) < \frac{\tilde{d}}{2\sqrt{n}}\right] \;\le\; \exp\left(-\frac{16\sqrt{n}\log n}{8\sqrt{n}}\right) \;=\; \frac{1}{n^2}.$$

□

Claim 5.4 *$\delta_H \ge \tilde{d}/(2\sqrt{n})$ with probability at least $1 - 1/n$.*

Proof: By Claim 5.3, the probability that *any* node v of H has $deg_H(v) < \tilde{d}/(2\sqrt{n})$ is at most $\frac{1}{n^2} \cdot |H| \le 1/n$. □

Claim 5.2 and Claim 5.4 together show that with probability at least $1 - \frac{1}{n} - \frac{1}{n^4} \ge 1 - \frac{2}{n}$, H has at most k vertices and has minimum degree at least $\tilde{d}/(2\sqrt{n})$. Therefore, with high probability, H provides a solution of DDDkS which is within a factor $1/(2\sqrt{n})$ of the optimal solution. This concludes the proof of the theorem. □

5.7 Conclusions

This chapter considered three Degree-Constrained Subgraph problems and studied their behavior in terms of approximation algorithms and hardness of approximation. Our main results and several interesting questions that remain open are discussed below.

We proved that the MDBCS$_d$ problem is not in Apx for any $d \geq 2$, and that if there is a polynomial time algorithm for MDBCS$_d$, $d \geq 2$, with a performance ratio of $2^{O(\sqrt{\log n})}$, then NP $\subseteq$ DTIME($2^{O(\log^5 n)}$). We provided a deterministic approximation algorithm with ratio $\min\{m/\log n,\ nd/(2\log n)\}$ (resp. $\min\{n/2,\ m/d\}$) for general unweighted (resp. weighted) graphs. Finally, we gave a constant-factor approximation when the input graph has a low-degree spanning tree. Closing the huge gap between the hardness bound and the approximation ratio of our algorithm looks like a promising research direction.

We proved that the MSMD$_d$ problem is not in Apx for any $d \geq 3$. It would be interesting to strengthen this hardness result using the power of the PCP theorem. On the positive side, we gave an $O(n/\log n)$-approximation algorithm for the class of graphs excluding a fixed graph H as a minor. Finding an approximation algorithm for MSMD$_d$ in general graphs seems to be a challenging open problem. It seems that MSMD$_d$ remains hard even for proper minor-closed classes of graphs.

We provided a $O(n^\delta)$-approximation algorithm for the DDDkS problem, for some universal constant $\delta < 1/3$. It would be interesting to provide hardness results complementing this approximation algorithm. Another avenue for further research could be to consider a mixed version between DDDkS and MSMD$_d$, that would result in a two-criteria optimization problem. Namely, given a graph G, the goal would be to maximize the minimum degree while minimizing the size of the subgraph, both parameters being subject to a lower and an upper bound, respectively.

Chapter 6

Parameterized Complexity of Finding Degree-constrained Subgraphs

In this chapter we study the parameterized complexity of problem of finding degree-constrained subgraphs, taking as the parameter the number of vertices of the desired subgraph. Namely, given two positive integers d and k, we study the problem of finding a d-regular (induced or not) subgraph with at most k vertices and the problem of finding a subgraph with at most k vertices and of minimum degree at least d.

We first show that both problems are fixed-parameter intractable in general graphs. More precisely, we prove that the first problem is $W[1]$-hard using a reduction from Multi-Color Clique. The hardness of the second problem follows from an easy extension of an already know result. We then provide explicit fixed-parameter tractable (FPT) algorithms to solve both problems in graphs with bounded local treewidth and graphs with excluded minors, using a dynamic programming approach. In particular, the problems become fixed-parameter tractable in planar graphs, graphs of bounded genus, and graphs with bounded maximum degree.

Keywords: parameterized complexity, degree-constrained subgraph, $W[1]$-hardness, dynamic programming, excluded minors.

6.1 Introduction

As discussed in Section III.1 (page 117), problems of finding subgraphs with certain degree constraints are well studied both algorithmically and combinatorially, and have a number of applications in network design [C6, 112, 135, 152, 153]. In this chapter we consider two natural such problems: finding a small regular (induced or not) subgraph and finding a

small subgraph with given minimum degree. We focus on these problems in Sections 6.1.1 and 6.1.2, respectively.

6.1.1 Finding a small regular subgraph

The complexity of finding regular graphs as well as regular (induced) subgraphs has been intensively studied in the literature [60, 64, 68, 73, 132, 161, 164, 185]. One of the first problems of this kind was stated by Garey and Johnson: Cubic Subgraph, that is, the problem of deciding whether a given graph contains a 3-regular subgraph, is NP-complete [73]. More generally, the problem of deciding whether a given graph contains a d-regular subgraph for any fixed degree $d \geq 3$ is NP-complete on general graphs [68] as well as on planar graphs [185] (where in the latter case only $d = 4$ and $d = 5$ were considered, since any planar graph contains a vertex of degree at most 5). Note that this problem is clearly polynomial-time solvable for $d \leq 2$. If the regular subgraph is required to be induced, Cardoso *et al.* proved that finding a maximum cardinality d-regular induced subgraph is NP-complete for any fixed integer $d \geq 0$ [64] (for $d = 0$ and $d = 1$ the problem corresponds to Maximum Independent Set and Maximum Induced Matching, respectively).

Concerning parameterized complexity of finding regular subgraphs, Moser and Thilikos proved that the following problem is $W[1]$-hard for every fixed integer $d \geq 0$ [164]:

$\geq k$-size d-Regular Induced Subgraph
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Question: Does there exist a subset $S \subseteq V$, with $\|S\| \geq k$, such that $G[S]$ is d-regular?

On the other hand, the authors proved that the following problem (which can be seen as the dual of the above one) is NP-complete but has a problem kernel of size $O(kd(k+d)^2)$ for $d \geq 1$ [164]:

$\leq k$-Almost d-Regular Graph
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Question: Does there exist a subset $S \subseteq V$, with $\|S\| \leq k$, such that $G[V \setminus S]$ is d-regular?

Mathieson and Szeider studied in [161] variants and generalizations of the problem of finding a d-regular subgraph (for $d \geq 3$) in a given graph by deleting at most k vertices. In particular, they answered a question of [164], proving that the $\leq k$-Almost d-Regular Graph problem (as well as some variants) becomes $W[1]$-hard when parameterized only by k (that is, it is unlikely that there exists an algorithm to solve it in time $f(k) \cdot n^{O(1)}$, where $n = |V(G)|$ and f is a function independent of n and d).

Given two integers d and k, it is also natural to ask for the existence of an induced d-regular graph with at most k vertices. The corresponding parameterized problem is defined as follows.

≤ k-SIZE d-REGULAR INDUCED SUBGRAPH (kdRIS)
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Question: Does there exist a subset $S \subseteq V$, with $|S| \leq k$, such that $G[S]$ is d-regular?

Note that the complexity of ≤ k-SIZE d-REGULAR INDUCED SUBGRAPH does not follow directly from the complexity of ≥ k-SIZE d-REGULAR INDUCED SUBGRAPH as, for instance, the approximability of the problems of finding a densest subgraph on *at least* k vertices or on *at most* k vertices are significantly different [36].

In general, a graph may not contain an induced d-regular subgraph on at most k vertices, while containing a non-induced d-regular subgraph on at most k vertices. This observation leads to the following problem:

≤ k-SIZE d-REGULAR SUBGRAPH (kdRS)
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Question: Does there exist a d-regular subgraph $H \subseteq G$, with $|V(H)| \leq k$?

Observe that ≤ k-SIZE d-REGULAR SUBGRAPH could a priori be easier than its corresponding induced version, as it happens for the MAXIMUM MATCHING (which is in P) and the MAXIMUM INDUCED MATCHING (which is NP-hard) problems.

To the best of our knowledge, the two parameterized problems defined above have not been considered in the literature. We prove in Section 6.2 that both problems are $W[1]$-hard for every fixed $d \geq 3$, by reduction from MULTI-COLOR CLIQUE.

6.1.2 Finding a small subgraph with given minimum degree

For a finite, simple, and undirected graph $G = (V, E)$ and $d \in \mathbb{N}$, the d-girth $g_d(G)$ of G is the minimum order of an induced subgraph of G of minimum degree at least d. The notion of d-girth was proposed and studied by Erdős *et al.* [109, 110] and Bollobás and Brightwell [59]. It generalizes the usual girth, the length of a shortest cycle, which coincides with the 2-girth. (This is indeed true because every induced subgraph of minimum degree at least two contains a cycle.) Combinatorial bounds on the d-girth can also be found in [53, 151]. The corresponding optimization problem is exactly the MSMD$_d$ problem defined in Chapter 5 (page 125). From the parameterized complexity point of view, it is natural to introduce a parameter $k \in \mathbb{N}$ and ask for the existence of a subgraph with at most k vertices and with minimum degree at least d. The problem can be formally defined as follows.

≤ k-SIZE SUBGRAPH OF MINIMUM DEGREE ≥ d (kSMDd)
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Question: Does there exist a subset $S \subseteq V$, with $|S| \leq k$, such that $G[S]$ has minimum degree at least d?

Note that the case $d = 2$ in in P, as discussed above. The special case of $d = 4$ appears in the book of Downey and Fellows [101], where it is stated that kSMD4 is $W[1]$-hard. From this result, it is easy to prove that kSMDd is $W[1]$-hard for every fixed $d \geq 4$ (see Section 6.2). The case $d = 3$ remains open. Note that in the kSMDd problem we can assume without loss of generality that we are looking for the existence of an induced subgraph, since we only require the vertices to have degree at least d.

6.1.3 Presentation of the results

We do a thorough study of the kdRS, the kdRIS, and the kSMDd problems in the realm of parameterized complexity. Some basic background of parameterized complexity can be found in Section I.2.3. Our results can be classified into two categories:

General graphs: We show in Section 6.2 that kdRS is not fixed-parameter tractable by showing it to be $W[1]$-hard for any $d \geq 3$ in general graphs. We will see that the graph constructed in our reduction implies also the $W[1]$-hardness of kdRIS. In general, parameterized reductions are quite stringent because of parameter-preserving requirements of the reduction, and require some technical care. Our reduction is based on a new methodology emerging in parameterized complexity, called *multi-color clique edge representation*. This has proved to be useful in showing various problems to be $W[1]$-hard recently [70]. We first spell out step by step the procedure to use this methodology, which can be used as a template for future purposes. Then we adapt this methodology to the reduction for the kSMDd problem. Our reduction is robust, in the sense that similar problems can be shown to be $W[1]$-hard with minor modifications. The hardness of kSMDd for $d \geq 4$ follows from an easy extension of a result of [101].

Graphs with bounded local treewidth and graphs with excluded minors: Both the kSMDd and kdRS problems can be easily defined in first-order logic, where the formula only depends on k and d, both being bounded by the parameter. Frick and Grohe [126] have shown that first-order definable properties of graph classes of bounded local treewidth can be decided in time $n^{1+1/k}$ for all k, in particular in time n^2, and first-order model checking is FPT on M-minor-free graphs. This immediately gives us the *classification result* that both problems are FPT for $d \geq 3$ in graphs with bounded local treewidth and graphs excluding a fixed graph M as a minor. These classification results can be generalized to a larger class of graphs, namely graphs locally excluding a fixed graph M as a minor, by a recent result of Dawar, Grohe and Kreutzer [87]. These results are by nature very general and can involve huge coefficients (dependence on k). A natural problem arising in this context is then the design of an explicit algorithm for kSMDd for $d \geq 3$ in these graph classes with explicit time complexity, faster than the one coming from the meta-theorem of Frick and Grohe. In Section 6.3, we provide a fast and explicit algorithms for kSMDd, $d \geq 3$, in graphs with bounded local treewidth and graphs excluding a fixed graph M as a minor. For the sake of simplicity, we present the algorithm for the kSMDd problems, but the same algorithms can be applied to the kdRS problem, with the same time bounds. Our algorithms use standard dynamic programming over graphs with bounded treewidth and

a few results concerning the clique decomposition of M-minor-free graphs developed by Robertson and Seymour in their graph minor theory [177]. A set of non-trivial observations allow to get improvements in the time complexity of the algorithms. We note that our dynamic programming over graphs with bounded treewidth is also generic and can handle variations on degree-constrained subgraph problems with simple changes.

6.2 Fixed-Parameter In-tractability Results

As mentioned in the introduction, kSMDd is known to be $W[1]$-hard for $d = 4$ [101]. It can be easily proved that kSMDd is $W[1]$-hard for every $d \geq 4$, by reducing kSMDd to kSMD$d+1$.

Indeed, let G be an instance of kSMDd, with parameter k. We construct an instance G' of kSMD$d+1$ from G by adding a vertex u and connecting it to all the vertices of G. We set the parameter to $k+1$. If there is a subset of vertices $S \subseteq V(G)$ of size at most k and with minimum degree at least d, then $S \cup \{u\}$ is a solution to kSMD$d+1$ in G' (the degree of u is also at least $d+1$ since we can assume that $k \geq d+1$). Conversely, if there is a subset of vertices $S \subseteq V(G')$ of size at most $k+1$ and with minimum degree at least $d+1$, we construct a solution to kSMDd in G as follows.

- if $u \in S$, then $S \setminus \{u\}$ is a solution in G.
- otherwise, if $u \notin S$, let v be an arbitrary vertex in S. Then any connected component of the subgraph induced by $S \setminus \{v\}$ is a solution in G, since $|S \setminus \{v\}| \leq k$ and the degrees of the vertices in $S \setminus \{v\}$ have decreased by at most 1 after the removal of v.

In the remainder of this section we give a $W[1]$-hardness reduction for kdRS. The definition of a parameterized reduction can be found in Section I.2.3 (page 18). Our reduction is from Multi-Color Clique, which is known to be $W[1]$-complete by a simple reduction from the ordinary Clique [113], and is based on the methodology known as *multi-color edge representation*. The Multi-Color Clique problem is defined as follows.

Multi-color Clique
Input: An graph $G = (V, E)$, a positive integer k, and a proper k-coloring of $V(G)$.
Parameter: k.
Question: Does there exist a clique of size k in G consisting of exactly one vertex of each color?

Consider an instance $G = (V, E)$ of Multi-color Clique with its vertices colored with the set of colors $\{c_1, \cdots, c_k\}$. Let $V[c_i]$ denote the set of vertices of color c_i. For each edge $e = \{u, v\}$ of G, with $u \in V[c_i]$, $v \in V[c_j]$, and $i < j$, we first replace e with two arcs $e^f = (u, v)$ and $e^b = (v, u)$. By abuse of notation, we also call this digraph G. Let $E[c_i, c_j]$ be the set of arcs $e = (u, v)$, with $u \in V[c_i]$ and $v \in V[c_j]$, for $1 \leq i \neq j \leq k$. An arc $(u, v) \in E[c_i, c_j]$ is called *forward* (resp. *backward*) if $i < j$ (resp. $i > j$). We also assume that $|V[c_i]| = N$ for all i, and that $|E[c_i, c_j]| = M$ for all $i \neq j$, i.e., we assume that the color classes of G,

and also the arc sets between them, have uniform sizes. For a simple justification of this assumption, we can reduce Multi-color Clique to itself, taking the union of $k!$ disjoint copies of G, one for each permutation of the color sets.

In this methodology, the basic encoding bricks correspond to the arcs of G, which we call **arc gadgets**. We generally have three kinds of gadgets, which we call **selection**, **coherence**, and **match gadgets**. These are engineered together to get an overall reduction gadget for the problem. In an optimal solution to the problem (that is, a solution providing a Yes answer), the selection gadget ensures that *exactly one* arc gadget is selected among arc gadgets corresponding to arcs going from a color class $V[c_i]$ to another color class $V[c_j]$. For any color class $V[c_i]$, the coherence gadget ensures that the out-going arcs from $V[c_i]$, corresponding to the selected arc gadgets, have a common vertex in $V[c_i]$. That is, all the arcs corresponding to these selected arc gadgets *emanate from the same vertex in* $V[c_i]$. Finally, the match gadget ensures that if we have selected an arc gadget corresponding to an arc (u, v) from $V[c_i]$ to $V[c_j]$, then the arc gadget selected from $V[c_j]$ to $V[c_i]$ corresponds to (v, u). That is, *both of* e^f *and* e^b *are selected together.* In what follows, we show how to particularize this general strategy to obtain a reduction from Multi-color Clique to kdRS for $d \geq 3$. To simplify the presentation, we first describe our reduction for the case $d = 3$ (in Section 6.2.1) and then we describe the required modifications for the case $d \geq 4$ in Section 6.2.2.

6.2.1 $W[1]$-hardness for the cubic case

In this section we give in detail the construction of all the gadgets for $d = 3$. Recall that an arc $(u, v) \in E[c_i, c_j]$ is forward if $i < j$, and it is backward if $i > j$. We refer the reader to Figure 6.1 to get an idea of the construction.

Arc gadgets: For each arc $(u, v) \in E[c_i, c_j]$ with $i < j$ (resp. $i > j$) we have a cycle C_{e^f} (resp. C_{e^b}) of length $3 + 2(k-2) + 2$, with the set of vertices:

- *selection vertices:* e^f_{s1}, e^f_{s2}, and e^f_{s3} (resp. e^b_{s1}, e^b_{s2}, and e^b_{s3});
- *coherence vertices:* e^f_{ch1r}, e^f_{ch2r} (resp. e^b_{ch1r}, e^b_{ch2r}), for all $r \in \{1, \ldots, k\}$ and $r \neq i, j$; and
- *match vertices:* e^f_{m1} and e^f_{m2} (resp. e^b_{m1} and e^b_{m2}).

Selection gadgets: For each pair of indices i, j with $1 \leq i \neq j \leq k$, we add a new vertex A_{c_i,c_j}, and connect it to all the selection vertices of the cycles C_{e^f} if $i < j$ (resp. C_{e^b} if $i > j$) for all $e \in E[c_i, c_j]$. This gadget is called *forward selection gadget* (resp. *backward selection gadget*) if $i < j$ (resp. $i > j$), and it is denoted by $\mathcal{S}_{i,j}$.

That is, we have $k(k-1)$ clusters of gadgets: one gadget $\mathcal{S}_{i,j}$ for each set $E[c_i, c_j]$, for $1 \leq i \neq j \leq k$.

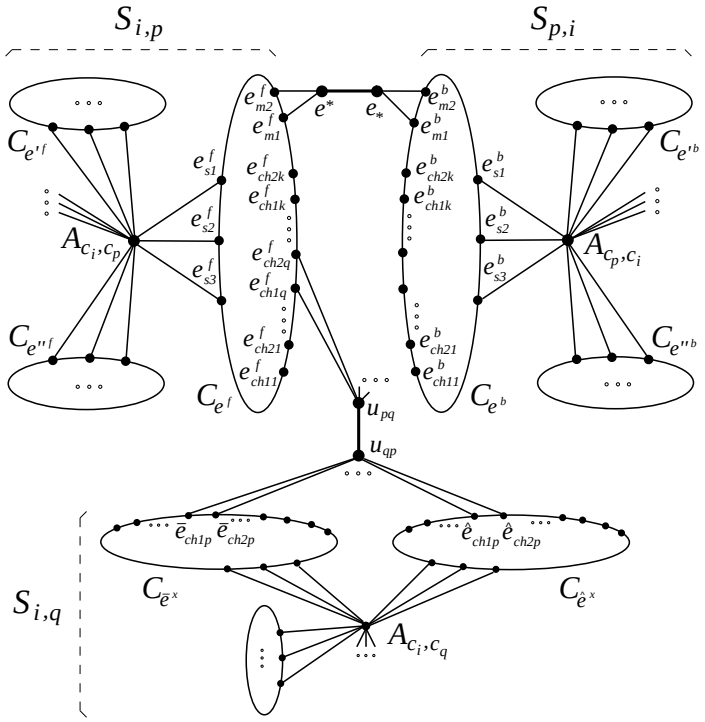

Figure 6.1: Gadgets used in the reduction of the proof of Theorem 6.1 (we suppose $i < p$).

Coherence gadgets: For each i, $1 \leq i \leq k$, let us consider all the selection gadgets of the form $\mathcal{S}_{i,p}$, $p \in \{1, \cdots, k\}$ and $p \neq i$. For any $u \in V[c_i]$, and any two indices $1 \leq p \neq q \leq k$, $p, q \neq i$, we add two new vertices u_{pq} and u_{qp}, and a new edge $\{u_{pq}, u_{qp}\}$. For every arc $e = (u, v) \in E[c_i, c_p]$, with $u \in V[c_i]$, we pick the cycle C_{e^x}, $x \in \{f, b\}$ depending on whether e is forward or backward, and add two edges of the form $\{e_{ch1q}, u_{pq}\}$ and $\{e_{ch2q}, u_{pq}\}$. Similarly, for an arc $e = (u, w) \in E[c_i, c_q]$, with $u \in V[c_i]$, we pick the cycle C_{e^x}, $x \in \{f, b\}$, and add two edges $\{e_{ch1p}, u_{qp}\}$ and $\{e_{ch2p}, u_{qp}\}$.

Match gadgets: For any pair of arcs $e^f = (u, v)$ and $e^b = (v, u)$, we consider the two cycles C_{e^f} and C_{e^b} corresponding to e^f and e^b. Now, we add two new vertices e^* and e_*, a *matching edge* $\{e^*, e_*\}$, and all the edges of the form $\{e^f_{m1}, e^*\}$, $\{e^f_{m2}, e^*\}$, $\{e^b_{m1}, e_*\}$ and $\{e^b_{m2}, e_*\}$ where e^f_{m1}, e^f_{m2} are match vertices on C_{e^f}, and e^b_{m1} , e^b_{m2} are match vertices on C_{e^b}.

This completes the construction of the gadgets, and the union of all of them defines the graph $\mathscr{G}_G$ depicted in Figure 6.1.

We now prove that this construction yields the reduction through a sequence of simple claims.

Claim 6.1 *Let G be an instance of* Multi-color Clique*, and $\mathscr{G}_G$ be the graph we constructed above. If G has a multi-colored k-clique, then $\mathscr{G}_G$ has a 3-regular subgraph of size $k' = (3k+1)k(k-1)$.*

Proof: Let ω be a multi-color clique of size k in G. For every edge $e \in E(\omega)$, select the corresponding cycles C_{e^f}, C_{e^b} in $\mathscr{G}_G$. Let us define S as follows.

$$S = \bigcup_{e\in\omega, x\in\{f,b\}} N_{\mathscr{G}_G}[V(C_{e^x})] \ .$$

It is straightforward to check that $\mathscr{G}_G[S]$ is a 3-regular subgraph of $\mathscr{G}_G$. To verify the size of $\mathscr{G}_G[S]$, note that we have $2 \cdot \binom{k}{2}$ cycles in $\mathscr{G}_G[S]$ and each of them contributes $(3k-1)$ vertices (this includes vertices on the cycle themselves). □

Claim 6.2 *Any 3-regular subgraph of $\mathscr{G}_G$ contains one of the cycles C_{e^x}, $x \in \{b, f\}$, corresponding to arc gadgets.*

Proof: Note that if such a subgraph of $\mathscr{G}_G$ intersects a cycle C_{e^x}, then it must contain all of its vertices. Further, if we remove all the vertices corresponding to arc gadgets in $\mathscr{G}_G$, then the remaining graph is a forest. These two facts together imply that any 3-regular subgraph of $\mathscr{G}(G)$ should intersect at least one cycle C_{e^x} corresponding to an arc gadget, hence it must contain C_{e^x}. □

Claim 6.3 *If $\mathscr{G}_G$ contains a 3-regular subgraph of size $k' = (3k+1)k(k-1)$, then G has a multi-colored k-clique.*

Proof: Let $H = G[S]$ be a 3-regular subgraph of size k'. Now, by Claim 6.2, S must contain all the vertices of a cycle corresponding to an arc gadget. Furthermore, notice that to ensure the degree condition in H, once we have a vertex of a cycle in S, all the vertices of this cycle and their neighbors are also in S. Without loss of generality, let C_{e^f} be this cycle, and suppose that it belongs to the gadget $\mathcal{S}_{i,j}$, i.e., $e \in E[c_i, c_j]$ and $i < j$. Notice that by construction, this forces some of the other vertices to belong also to S. Indeed, its match vertices force the cycle C_{e^b} of $\mathcal{S}_{j,i}$ to be in S. The coherence vertices of C_{e^f} force S to contain at least one cycle in $\mathcal{S}_{i,l}$, for all $l \in \{1, \cdots, k\}$, $l \neq i$. They in turn force S to contain at least one cycle from the remaining gadgets $\mathcal{S}_{p,q}$ for all $p \neq q \in \{1, \ldots, k\}$. The selection vertices of each such cycle in $\mathcal{S}_{p,q}$ force S to contain $A_{p,q}$. But because of our condition on the size of S ($|S| = k'$), we can select *exactly one* cycle gadget from each of the gadgets $\mathcal{S}_{p,q}$, $p \neq q \in \{1, 2, \cdots, k\}$. Let E' be the set of edges in $E(G)$ corresponding to arc gadgets selected in S. We claim that $G[V[E']]$ is a multi-color clique of size k in G. Here $V[E']$ is a subset of vertices of $V(G)$ containing the end points of the edges in E'. First of all, because of the match vertices, once e^f is in E', e^b is forced to be in E'. To conclude the proof we only need to ensure that all the edges from a particular color class emanate from the same vertex. But this is ensured by the restriction on the size of S and the presence of coherence vertices on the cycles selected in S from $\mathcal{S}_{p,q}$, $p \neq q \in \{1, 2, \cdots, k\}$. To see this, let us take two arcs $e = (u, v) \in (E[c_i, c_p] \cap E')$ and $e' = (u', w) \in (E[c_i, c_q] \cap E')$. Now the four vertices u_{pq}, u_{qp}, u'_{pq}, and u'_{qp} belong to S. If u is different from u', then S has at least two elements more than the expected size k', which contradicts the condition on the size of S. All these facts together imply that $G[V(E')]$ forms a multi-colored k-clique in the original graph G. □

Claims 6.1 and 6.3 together yield the following theorem:

Theorem 6.1 *k*3RS *is* W[1]*-hard.*

We shall see in the next section that the proof of the Theorem 6.1 can be generalized to larger values of d. Note that the 3-regular subgraph constructed in the proof of Theorem 6.1 is a 3-regular *induced* subgraphs, so our proof implies the following corollary.

Corollary 6.1 *k*3RIS *is* W[1]*-hard.*

6.2.2 *W*[1]-hardness for higher degrees

In this section we generalize the reduction given in Section 6.2 for $d \geq 4$. The main idea is to change the role of the cycles C_e by $(d-1)$-regular graphs of appropriate size. We show below all the necessary changes in the construction of the gadgets to ensure that the proof for $d = 3$ works for $d \geq 4$.

Arc gadgets for $d \geq 4$***:*** Let us take C to be a $(d-1)$-regular graph of size $(d-1)+(d-1)(k-2)+d$, if it exists (that is, if $(d-1)$ is even or k is odd). If such a graph does not exist, we take a graph of size $(d-1)+(d-1)(k+2)+d+1$ and with regular degree $d-1$ on the set C of $(d-1)+(d-1)(k+2)+d$ vertices and degree d on the last vertex v. As before, we replace each edge e with two arcs e^f and e^b. For each arc $e^x \in E[c_i, c_j]$, we add a copy of C, that we call C_{e^x}, with the following vertex set:

- *selection vertices:* $e^x_{s1}, e^x_{s2}, \cdots, e^x_{sd}$;
- *coherence vertices:* $e^x_{ch1r}, \cdots, e^x_{ch(d-1)r}$, for all $r \in \{1, \ldots, k\}$, $r \neq i, j$; and
- *match vertices:* $e^x_{m1}, \cdots, e^x_{m(d-1)}$.

Selection gadgets for $d \geq 4$***:*** Without loss of generality suppose that $x = f$. As before, we add a vertex A_{c_i,c_j}, and for every arc $e^f \in E[c_i, c_j]$ we add all the edges from A_{c_i,c_j} to all the selection vertices of the graph C_{e^f}. We call this gadget $\mathcal{S}_{i,j}$.

Coherence gadgets for $d \geq 4$***:*** Fix an i, $1 \leq i \leq k$. Let us consider all the selection gadgets of the form $\mathcal{S}_{i,p}$, $p \in \{1, \cdots, k\}$ and $p \neq i$. For any $u \in V[c_i]$, and any two indices $p \neq q \leq k$, $p, q \neq i$, we add a new edge $\{u_{pq}, u_{qp}\}$. For every arc $e = (u, v) \in E[c_i, c_p]$, with $u \in V[c_i]$, we pick the graph C_{e^x}, $x \in \{f, b\}$, depending on whether e is forward or backward, and add $d-1$ edges of the form $\{e_{ch1q}, u_{pq}\}, \{e_{ch2q}, u_{pq}\}, \ldots, \{e_{ch(d-1)q}, u_{pq}\}$. Similarly, for an arc $e = (u, w) \in E[c_i, c_q]$, with $u \in V[c_i]$, we pick the graph C_{e^x}, $x \in \{f, b\}$, and add $d-1$ edges of the form $\{e_{ch1p}, u_{qp}\}, \ldots, \{e_{ch(d-1)p}, u_{qp}\}$.

Match gadgets for $d \geq 4$: For the two arcs $e^f = (u, v)$ and $e^b = (v, u)$, we consider the two graphs C_{e^f} and C_{e^b} corresponding to e^f and e^b. Now we add a matching edge $\{e^*, e_*\}$ and add all the edges of the form $\{e^f_{m1}, e^*\}, \ldots, \{e^f_{m(d-1)}, e^*\}$ and $\{e^b_{m1}, e_*\}, \ldots, \{e^b_{m1}, e_*\}$, where e^f_{mi}, e^b_{mi} are match vertices of C_{e^f} and of C_{e^b}, respectively.

This completes the construction of the gadgets, and the union of all of them defines the graph $\mathscr{G}_G$. It is not hard to see that a proof similar to that of Theorem 6.1 shows that G, an instance of multi-color clique, has a multi-colored clique of size k if and only if $\mathscr{G}_G$ has a d-regular subgraph of size $k' = dk + 1$. We have the following theorem.

Theorem 6.2 *kdRS is W[1]-hard for all $d \geq 3$.*

Notice that again the d-regular subgraph constructed in the proof of Theorem 6.2 turn out to be an induced subgraph of regular degree d in $\mathcal{G}_G$. As a consequence we obtain the following corollary:

Corollary 6.2 *kdRIS is W[1]-hard for all $d \geq 3$.*

6.3 FPT Algorithms for Graphs with Bounded Local Treewidth and Graphs with Excluded Minors

In this section, we provide explicit (and fast) algorithms for kSMDd, $d \geq 3$, in graphs with bounded local treewidth (Section 6.3.1) and in graphs excluding a fixed graph M as a minor (Section 6.3.2). We first provide the necessary background.

The definition of treewidth (see page 15) can be generalized to take into account the local properties of G, and this is called *local treewidth*. To define it formally, we first need to define the r-neighborhood of vertices of G. The *distance* $d_G(u, v)$ between two vertices u and v of G is the length of a shortest path in G from u to v. For $r \geq 1$, a *r-neighborhood* of a vertex $v \in V$ is defined as $N^r_G(v) = \{u \in V \mid d_G(v, u) \leq r\}$.

The *local treewidth* of a graph G is a function $ltw^G : \mathbb{N} \to \mathbb{N}$ which associates to every integer $r \in \mathbb{N}$ the maximum treewidth of an r-neighborhood of vertices of G, i.e.,

$$ltw^G(r) = \max_{v \in V(G)} \{tw(G[N^r_G(v)])\}.$$

A graph class $\mathscr{G}$ has *bounded local treewidth* if there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that for each graph $G \in \mathscr{G}$ and for each integer $r \in \mathbb{N}$, we have $ltw^G(r) \leq f(r)$. For a given function $f : \mathbb{N} \to \mathbb{N}$, $\mathscr{G}_f$ is the class of all graphs G of local tree-width at most f, i.e., such that $ltw^G(r) \leq f(r)$ for every $r \in \mathbb{N}$. We refer to [106] and [137] for more details.

We now provide the basics to understand the structure of the classes of graphs excluding a fixed graph as a minor.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two disjoint graphs, and $k \geq 0$ an integer. For $i = 1, 2$, let $W_i \subseteq V_i$ form a clique of size h and let G'_i be the graph obtained from G_i by removing a set of edges (possibly empty) from the clique $G_i[W_i]$. Let $F : W_1 \to W_2$ be

a bijection between W_1 and W_2. The *h-clique sum* or the *h-sum* of G_1 and G_2, denoted by $G_1 \oplus_{h,F} G_2$, or simply $G_1 \oplus G_2$ if there is no confusion, is the graph obtained by taking the union of G_1' and G_2' by identifying $w \in W_1$ with $F(w) \in W_2$, and by removing all the multiple edges. The image of the vertices of W_1 and W_2 in $G_i \oplus G_2$ is called the *join* of the sum.

Note that $\oplus$ is not well defined; different choices of G_i' and the bijection F can give different clique sums. A sequence of h-sums, not necessarily unique, which result in a graph G, is called a *clique sum decomposition* or, simply, a *clique decomposition* of G.

Let Σ be a surface with boundary cycles $C_1, \ldots, C_h$. A graph G is *h-nearly embeddable* in Σ, if G has a subset X of vertices of size at most h, called *apices*, such that there are (possibly empty) subgraphs $G_0, \ldots, G_h$ of $G \setminus X$ such that

1. $G \setminus X = G_0 \cup \cdots \cup G_h$;
2. G_0 is embeddable in Σ (we fix an embedding of G_0);
3. $G_1, \ldots, G_h$ are pairwise disjoint;
4. For $1 \leq \cdots \leq h$, let $U_i := \{u_{i_1}, \ldots, u_{i_{m_i}}\} = V(G_0) \cap V(G_i)$, G_i has a path-decomposition $(\{B_{ij}\},\ 1 \leq j \leq m_i)$ of width at most h such that
   (a) for $1 \leq i \leq h$ and for $1 \leq j \leq m_i$ we have $u_j \in B_{ij}$; and
   (b) for $1 \leq i \leq h$, we have $V(G_0) \cap C_i = \{u_{i_1}, \ldots, u_{i_{m_i}}\}$ and the points $u_{i_1}, \ldots, u_{i_{m_i}}$ appear on C_i in this order (either walking through the cycles clockwise or counterclockwise).

6.3.1 Graphs with bounded local treewidth

In order to prove our results, we need the following lemma, which gives the time complexity of finding a smallest induced subgraph of degree at least d in graphs with bounded treewidth.

Lemma 6.1 *Let G be a graph on n vertices with a tree-decomposition of width at most t, and let d be a positive integer. Then in time $O((d+1)^t(t+1)^{d^2}n)$ we can decide whether there exists an induced subgraph of degree at least d in G and, if such a subgraph exists, find one of the smallest size.*

Proof: Let $(T, \mathcal{X})$ be the given tree-decomposition. We assume that T is a rooted tree, and that the decomposition is *nice*, which means the following:

- Each node has at most two children;
- For every node t with exactly two children t_1 and t_2, $X_t = X_{t_1} = X_{t_2}$;
- For every node t with exactly one child s, either $X_t \subset X_s$ and $|X_s| = |X_t| + 1$, or $X_s \subset X_t$ and $|X_t| = |X_s| + 1$.

Note that such a decomposition always exists and can be found in linear time, and in fact we may assume that $|V(T)| = O(n)$. As usual in algorithms based on tree decompositions,

we employ a dynamic programming approach based on this decomposition, which at the end either produces a connected subgraph of G of minimum degree at least d and of size at most k, or decides that G does not have any such subgraph.

As the tree decomposition is rooted, we can speak of the subgraph defined by the subtree rooted at node i. More precisely, for any node i of T, let Y_i be the set of all vertices that appear either in X_i or in X_j for some descendant j of i. Denote by $G[Y_i]$ the graph induced by the nodes in Y_i.

Note that if i is a node in the tree and j_1 and j_2 are two children, then Y_{j_1} and Y_{j_2} are disjoint except for vertices in X_i, i.e., $Y_{j_1} \cap Y_{j_2} = X_i$. A $\mathcal{P}$-coloring of the vertices in X_i, for the palette $\mathcal{P} = \{0, 1, \ldots, d\}$, is a function $c_i : X_i \to \mathcal{P}$. The *support* of c is $supp(c) = \{v \in X_i \mid c(v) \neq 0\}$.

For any such $\mathcal{P}$-coloring c of vertices in X_i, let $a(i, c)$ be the minimum size of an induced subgraph $H(i, c)$ of $G[Y_i]$, which has degree $c(v)$ for every $v \in X_i$ with $c(v) \neq d$, and degree at least d on its other vertices. Note that $H(i, c) \cap X_i = supp(c)$. If such a subgraph does not exist, we define $a(i, c) = +\infty$.

We develop recursive formulas for $a(i, c)$. In the base case, i is a leaf of the tree decomposition. Hence $Y_i = X_i$. The size of the minimum induced subgraph with prescribed degrees is exactly $|supp(c)|$ if $G[supp(c)]$ satisfies the degree conditions, and is $+\infty$ if it does not.

In the recursive case, node i has at least one child. We distinguish between three cases, depending on the size of the bag of i and its number of children.

Case (1): i has only one child j and $X_i \subset X_j$.
Then $|X_j| = |X_i| + 1$ and $X_i = X_j \setminus \{v\}$ for some vertex v. Also, $Y_i = Y_j$, since X_i does not add any new vertices. Consider a coloring $c : X_i \to \mathcal{P}$. Consider the two colorings $c_0 : X_j \to \mathcal{P}$ and $c_1 : X_j \to \mathcal{P}$ of X_j, defined as follows: $c_0 = c_1 = c$ on X_i, and $c_0(v) = 0$, $c_1(v) = d$. Then we let $a(i, c) = \min\{a(j, c_0), a(j, c_1)\}$.

Case (2): i has only one child j and $X_j \subset X_i$.
Then $|X_j| = |X_i| - 1$ and $X_j = X_i \setminus \{v\}$ for some vertex v. Also, $Y_j = Y_i \setminus \{v\}$. Let c be a coloring of X_i. It is clear that the only neighbors of v in $G[Y_i]$ are already in X_i.

- If $c(v) \geq 1$, for any collection $\mathcal{A}$ of $c(v)$ edges in $G[X_i]$ connecting v to vertices $v_1, \ldots, v_{c(v)}$, with $c(v_i) \geq 1$ (note that such a collection may not exist at all), we consider the coloring $c_{\mathcal{A}}$ of X_j as follows: $c_{\mathcal{A}}(v_i) = c(v_i) - 1$ for any $1 \leq i \leq c(v)$, and $c_{\mathcal{A}}(w) = c(w)$ for any other vertex w. Then we define

$$a(i, c) = \min_{\mathcal{A}}\{a(j, c_{\mathcal{A}})\} + 1 \ .$$

- If $c(v) = 0$, we simply define $a(i, c) = a(j, c)$.

Note that there are at most $(t + 1)^{d+1}$ choices for such a collection $\mathcal{A}$.

Case (3): i has two children j_1 and j_2.
Then $X_i = X_{j_1} = X_{j_2}$. Let c be a coloring of X_i, then $supp(c) \subset X_i$ is part of the subgraph we are looking for. For any vertex $v \in X_i$, calculate the degree $deg_{G[X_i]}(v)$. Suppose that v has degree d_1^v, d_2^v in $H \cap G[Y_{j_1}], H \cap G[Y_{j_2}]$ (H is the subgraph we are looking for). These

degree sequences should guarantee the degree condition on v imposed by the coloring c. In other words, if $c(v) \leq d-1$ then we should have $d_1^v + d_2^v - d_{G[X_i]} = c(v)$, and if $c(v) = d$, then $d_1^v + d_2^v - d_{G[X_i]} \geq d$. Every such sequence $\mathcal{D} = \{d_1^v, d_2^v \mid v \in X_i\}$ on vertices of X_i determines two colorings $c_1^{\mathcal{D}}$ and $c_2^{\mathcal{D}}$ of X_{j_1} and X_{j_2} respectively. For each such pair of colorings, let H_1 and H_2 be the minimum subgraphs with these degree constraints in $G[Y_{j_1}]$ and $G[Y_{j_2}]$ respectively. Then $H_1 \cup H_2$ satisfies the degree constraints imposed by c. We define

$$a(i,c) = \min_{\mathcal{D}}\{|H| \mid H = H_1 \cup H_2\}$$

for all degree distributions as above. For every vertex we have at most d^2 possible degree choices for d_1^v and d_2^v. We have also $|X_i| \leq t+1$. This implies that the minimum is taken over at most $(t+1)^{d^2}$ colorings.

As the size of our tree-decomposition is linear on n, we can determine all the values $a(i,c)$ for every $i \in V(T)$ and every coloring of X_i in time linear in n. Now return the minimum value of $a(i,c)$ computed for all colorings c, for values in the set $\{0,d\}$ assigning at least one non-zero value. The time dependence on t follows from the size of the bags and the choices made using the colorings. □

Lemma 6.1 leads to the following theorem:

Theorem 6.3 *For any $d \geq 3$ and any function $f : \mathbb{N} \to \mathbb{N}$, kSMDd is fixed-parameter tractable on $\mathcal{G}_f$. Furthermore, the algorithm runs in time $O((d+1)^{f(2k)}(f(2k)+1)^{d^2}n^2)$.*

Proof: Let $G = (V,E)$ be a graph in $\mathcal{G}_f$, that is G has bounded local treewidth and the bound is given by the function f. We first notice that if there exists an induced subgraph $H \subseteq G$ of size at most k and degree at least d, then H can be supposed to be connected. Secondly, if we know a vertex v of H, then H is contained in $N_G^k[v]$, which has diameter at most $2k$. Hence there exists the desired H if and only if there exists $v \in V$ such that H is contained in $N_G^k[v]$. To solve the problem, for each $v \in V$, we find a tree-decomposition of $N_G^k[v]$ of width at most $f(2k)$ in time polynomial in n, and then run the algorithm of Lemma 6.1. □

The function $f(k)$ is known to be $3k$, $C_g gk$, and $b(b-1)^{k-1}$ for planar graphs, graphs of genus g, and graphs of degree at most b, respectively [106, 137]. Here C_g is a constant depending only on the genus g of the graph. As an easy corollary of Theorem 6.3, we have the following:

Corollary 6.3 *kSMDd can be solved in $O((d+1)^{6k}(6k+1)^{d^2}n^2)$, $O((d+1)^{2C_g gk}(2C_g gk+1)^{d^2}n^2)$ and $O((d+1)^{2b(b-1)^{k-1}}(2b(b-1)^{k-1}+1)^{d^2}n^2)$ time in planar graphs, graphs of genus g, and graphs of degree at most b, respectively.*

6.3.2 M-minor-free graphs

In this section, we consider the class of M-minor-free graphs. We need the following theorem of Robertson and Seymour [177] (see also Demaine et al. in [90] for an algorithmic version).

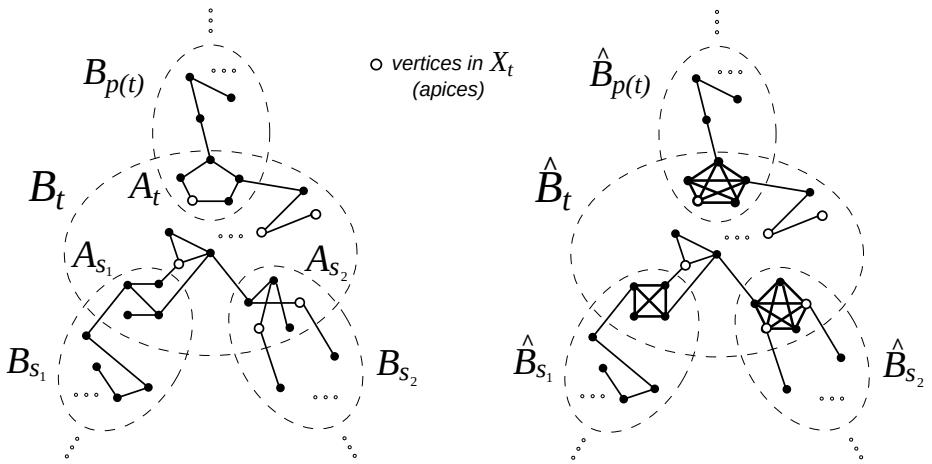

Figure 6.2: Tree-decomposition of a minor-free graph. The vertices in X_t (i.e., the *apices*) are depicted by $\circ$. Note that B_{s_1} and B_{s_2} could have non-empty intersection (in B_t).

Theorem 6.4 ([90, 177]) *For every graph M, there exists an integer h, depending only on the size of M, such that every graph excluding M as a minor can be obtained by clique sums of order at most h from graphs that can be h-nearly embedded in a surface Σ in which M cannot be embedded. Furthermore, such a clique decomposition can be found in polynomial time.*

Let G be an M-minor-free graph, and let $(T, \mathcal{B} = \{B_t\})$ be a clique decomposition of G given by Theorem 6.4. We suppose in addition that T is rooted at a given vertex $r \in V(G)$. We define $A_t := B_t \cap B_{p(t)}$ where $p(t)$ is the unique parent of the vertex t in T, and $A_r = \emptyset$. Let $\hat{B}_t$ be the graph obtained from B_t by adding all the possible edges between the vertices of A_t and also between the vertices of A_s, for each child s of t. In this way, A_t and A_s's will induce cliques in $\hat{B}_t$ (see Figure 6.2). In addition, G becomes an h-clique sum of the graphs $\hat{B}_t$ according to the above tree T where each $\hat{B}_t$ is h-nearly embeddable in a surface Σ in which M cannot be embedded. Let X_t be the set of apices of $\hat{B}_t$; we have $|X_t| \leq h$ and $\hat{B}_t \setminus X_t$ has linear local treewidth. We denote by G_t the subgraph induced by all the vertices of $B_t \cup \bigcup_s B_s$, for s ranging over all descendants of t in T.

In order to simplify the presentation, in what follows, we will restrict ourselves to the case $d = 3$, but it is quite straightforward to check that the proof extends to all $d \geq 3$. Recall that we are looking for a subset of vertices S, of size at most k, which induces a graph $H = G[S]$ of minimum degree at least three.

Our algorithm consists of two levels of dynamic programming. The top level of dynamic programming runs over the clique decomposition, and within each subproblem of this dynamic programming, we focus on the induced subgraph of the vertices in B_t. Our first level of dynamic programming computes the size of a smallest subgraph of G_t, complying with degree constraints on the vertices of A_t. These constraints, as before, represent the

degree of each vertex of A_t in the subgraph $H_t := G_t[S_t]$, i.e., the *trace* of H in G_t, where $S_t = S \cap V(G_t)$. This two-level dynamic programming requires a combinatorial bound on the treewidth as a function of the parameter k for each of the B_t's (after removing the apices X_t from B_t). The next two lemmas are used later to obtain this combinatorial bound.

Lemma 6.2 *Let $H = G[S]$ be a connected induced subgraph of G. Then the subgraph $\hat{B}_t[S \cap B_t]$ is connected.*

The proof of Lemma 6.2 easily follows from the properties of a tree-decomposition and the fact that A_t and A_s's are cliques in $\hat{B}_t$, for s a child of t in T.

Lemma 6.3 *Let $H = G[S]$ be a smallest connected subgraph of G of minimum degree at least three. Then the subgraph $\hat{B}_t[S_t \cap B_t \setminus X_t]$ has at most $3h + 1$ connected components, where h is the integer given by Theorem 6.4.*

Proof: Let $C_1, \ldots, C_r$ be the connected components of $L := \hat{B}_t[S_t \cap B_t \setminus X_t]$. We want to prove that $r \leq 3h + 1$. Assume for the sake of a contradiction that $r > 3h + 1$. We will find another solution H' with size strictly smaller than H, which will contradict our assumption that H is of minimum size.

The graph H' is defined as follows. For each vertex $v \in X_t \cap S_t$, let

$$b_v := \min\{d_{H_t}(v), 3\}.$$

Then for each vertex $v \in X_t \cap S_t$, we choose at most b_v connected components of L, covering at least b_v neighbors of v in H_t. We also add the connected component containing all the vertices of $A_t \setminus X_t$ (recall that A_t induces a clique in $\hat{B}_t$). Let A be the union of all the vertices of these connected components. Since $|X_t| \leq h$, A has at most $3h + 1$ connected components. Also, since A_s induces a clique in $\hat{B}_t$, for each child s of t such that $A_s \cap A \neq \emptyset$, we have that $A_s \setminus X_t \subset A$. We define H' as follows.

$$H' := G\left[\left(\bigcup_{\{s \,:\, A_s \cap A \neq \emptyset\}} S_s\right) \cup ((X_t \cup A) \cap S_t) \cup (S \setminus S_t)\right].$$

Clearly, $H' \subseteq H$. We have that $|H'| < |H|$ because, assuming that $r > 3h+1$, there are some vertices of $H_t \subset H$ which are in some connected component C_i which does not intersect H'.

Thus, it just remains to prove that H' is indeed a solution of kSMD3, i.e., H' has minimum degree at least 3. We prove it using a sequence of four simple claims:

Claim 6.4 *The degree of each vertex $v \in (V(H') \cap X_t)$ is at least 3 in H'.*

Proof: This is because each such vertex v has degree at least b_v in H'_t. If $d_v < 3$, then v should be in A_t (if not, v has degree $d_v < 3$ in H, which is impossible), hence v is connected to at least $3 - d_v$ vertices in $S \setminus S_t$. But $S \setminus S_t$ is included in H', and so every vertex of $X_t \cap V(H')$ has degree at least 3 in H'. □

Claim 6.5 *The degree of each vertex in* $(H \setminus H_t)$ *is at least* 3 *in* H'.

Proof: This follows because $A_t \cap H \subset H'$. □

Claim 6.6 *The degree of each vertex in* A *is at least* 3 *in* H'.

Proof: Every vertex in A has the same degree in both H' and H. This is because A is the union of some connected components, and no vertex of A is connected to any other vertex in any other component. □

Claim 6.7 *Every other vertex of* H' *also has degree at least* 3.

Proof: To prove the claim we prove that the vertices of $H' \setminus (G[X_t] \cup (H \setminus H_t) \cup A)$ have degree at least 3 in H'. Remember that all these vertices are in some S_s, for some s such that A_s has a non-empty intersection with A. We claim that all these vertices have the same degree in both H and H'. To prove this, note that $H' \cap A_s = H \cap A_s$ for all such s. Indeed, $(A_s \setminus X_t) \subset A$, and so $A_s \subset (A \cup X_t)$. Let u be such a vertex. We can assume that $u \notin X_t$. If $u \in A_s$, then clearly $u \in A$, and we are done. If $u \in (S_s \setminus A)$, then every neighbor of u is in H_s. But $H_s \subset H'$, hence we are also done in this case. □

This concludes the proof of the lemma. □

We define a *coloring* of A_t to be a function $c : A_t \cap S \to \{0, 1, 2, 3\}$. For $i < 3$, $c(v) = i$ means that the vertex v has degree i in the subgraph H_t of G_t that we are looking for, and $c(v) = 3$ means that v has degree at least three in H_t. By $a(t, c)$ we denote the minimum size of a subgraph of G_t with the prescribed degrees in A_t according to c. We describe in what follows the different steps of our algorithm.

Recursively, starting from the leaves of T and moving towards the root, for each node $t \in V(T)$ and for every coloring c of A_t, we compute $a(t, c)$ from the values of $a(s, c)$, where s is a child of t, or we store $a(t, c) = +\infty$ if no such subgraph exists. The steps involved in computing $a(t, c)$ for a fixed coloring c are the following:

(i) We guess a subset $R_t \subseteq X_t \setminus A_t$ such that $R_t \subseteq S_t$. We have at most 2^h choices for R_t.

(ii) For each vertex v in R_t, we guess whether v is adjacent to a vertex of $B_t \setminus (R_t \cup A_t)$, i.e., we test all the 2-colorings $\gamma : R_t \to \{0, 1\}$; a coloring has the following meaning: $\gamma(v) = 1$ if and only if v is adjacent to a vertex of $B_t \setminus (R_t \cup A_t)$. The number of such colorings is at most 2^h. Let γ be a fixed coloring. For each of the vertices v in R_t with $\gamma(v) = 1$, we guess one vertex in $B_t \setminus (R_t \cup A_t)$, which we suppose to be in S_t. For each coloring γ, we have at most n^h choices for the new vertices which could be included in S_t. If a vertex has $\gamma(v) = 0$, it is not allowed to be adjacent to any vertex of B_t besides the vertices in $A_t \cup R_t$. Let D_t^γ be the chosen vertices at this level.

(iii) We remove now all the vertices of X_t from B_t. Lemma 6.3 ensures that the induced graph $\hat{B}_t[S_t \cap B_t \setminus X_t]$ has at most $3h+1$ connected components. We then choose these connected components of $\hat{B}_t[S_t \cap B_t \setminus X_t]$ by guessing a vertex from these connected components in $B_t \setminus X_t$. Since we need to choose at most $3h+1$ vertices this way, we have at most $(3h+1)n^{3h+1}$ new choices. Let these newly chosen vertices be F_t^γ and

$$R_t^\gamma = R_t \cup D_t^\gamma \cup F_t^\gamma \cup \{v \in A_t \setminus X_t \mid c(v) \neq 0\}.$$

Let G_t^* be the graph induced by the k-neighborhood (vertices at distance at most k) of all vertices of R_t^γ in $\hat{B}_t \setminus X_t$, i.e., $G_t^* = (\hat{B}_t \setminus X_t)[N^k(R_t^\gamma)]$.

(iv) Each connected component of G_t^* has diameter at most $2k$ in $\hat{B}_t \setminus X_t$. As $\hat{B}_t \setminus X_t$ has bounded local treewidth, this implies that G_t^* has treewidth bounded by a function of k. By the result of Demaine and Hajiaghayi [91], this function can be chosen to be linear.

(v) In this step, we first find a tree-decomposition $(\mathcal{T}_\gamma, \{U_p\})$ of G_t^*. Since $A_s \cap G_t^*$ is a clique, it appears in a bag of this tree-decomposition. Let p be the node representing this bag in $\mathcal{T}_\gamma$. We create now a new bag containing the vertices of $A_s \cap G_t^*$, and modify $\mathcal{T}_\gamma$ by adding a leaf connected to p which contains this new bag. With slight abuse of notation, we call this new decomposition $\mathcal{T}_\gamma$ and denote by s this distinguished leaf containing the bag $A_s \cap G_t^*$. We also add all the vertices of A_t to all the bags of this tree-decomposition, increasing the bag size by at most h. Now we apply a dynamic programming algorithm similar to the one we used for the bounded local treewidth case. Remember that for each child s of t, we have a leaf in this (new) decomposition with the bag $A_s \cap G_s^*$. The aim is to find an induced subgraph of minimum size which respects all the choices we have made earlier.

We start from the leaves of $\mathcal{T}_\gamma$ and move towards its root. At this point we have all the values of $a(s, c')$ for all possible colorings c' of A_s, where s is a child of t (because of the first level of dynamic programming). To compute $a(t, c)$ we apply the dynamic programming algorithm of Lemma 6.1 with the restriction that for each *distinguished* leaf s of this decomposition, we already have all the values $a(s, c)$ for all colorings of $A_s \cap G_s^*$ (we extend this coloring to all A_s by giving the zero values to the vertices of $A_s \setminus G_s^*$). Note that the only difference between this dynamic programming and the one of Lemma 6.1 is the way we initialize the leaves of the tree.

(vi) Among all the subgraphs we found in this way, we keep the minimum size of a subgraph with the degree constraint c on A_t. Let $a(t, c)$ be this minimum.

(vii) If for some vertex t and a coloring $c : A_t \to \{0, 3\}$, we have $1 \leq a(t, c) \leq k$, the algorithm return YES, meaning that the graph contains a subgraph of size at most k and minimum degree at least three. If not, we conclude that such a subgraph does not exist.

This completes the description of the algorithm. Now we discuss the time complexity of this algorithm. Let C_M be the constant determining the linear local treewidth of the surfaces in which M cannot be embedded. For each fixed coloring c, we need time $4^{C_M k}(C_M k+1)^9 n^{4h+1}$ to obtain $a(t, c)$, where $t \in T$. Since the number of colorings of each A_t is at most 4^h, and the size of the clique decomposition is $O(n)$, we get the following theorem:

Theorem 6.5 *Let $\mathcal{C}$ be the class of graphs with excluded minor M. Then, for any graph in $\mathcal{C}$, one can find an induced subgraph of size at most k with degree at least 3 in time $O(4^{O(k+h)}(O(k))^9 n^{O(1)})$, where the constants in the exponents depend only on M.*

Theorem 6.5 can be generalized to larger values of d with slight modifications. We have the following theorem:

Theorem 6.6 *Let $\mathcal{C}$ be a class of graphs with an excluded minor M. Then, for any graph in $\mathcal{C}$, one can find an induced subgraph of size at most k with degree at least d in time $O((d+1)^{O(k+h)}(O(k))^{d^2} n^{O(1)})$, where the constants in the exponents depend only on M.*

6.4 Conclusions

In this chapter we studied the parameterized complexity of the following two problems: given two positive integers d and k, finding a d-regular (induced or not) subgraph with at most k vertices, and finding a subgraph with at most k vertices and of minimum degree at least d.

We first showed that both problems are fixed-parameter intractable in general graphs. More precisely, we proved that the first problem is $W[1]$-hard using a reduction from Multi-Color Clique. The hardness of the second problem followed from an extension of an already know result. We then provided explicit fixed-parameter tractable (FPT) algorithms to solve both problems in graphs with bounded local treewidth and graphs with excluded minors, using a dynamic programming approach. These algorithms are considerably faster than those coming from the meta-theorem of Frick and Grohe [126] about problems definable in first-order logic over "locally tree-decomposable structures".

Finally, note that the parameterized tractability of the kSMDd problem for the case $d = 3$ remains open. We conjecture that:

Conjecture 6.1 kSMD3 *is* $W[1]$*-hard.*

Chapter 7

Subexponential Parameterized Algorithms on Planar Graphs

In this chapter we present subexponential parameterized algorithms on planar graphs for a family of problems that consist in, given a graph G, finding a connected (induced) subgraph H with bounded maximum degree, while maximizing the number of edges (or vertices) of H. These problems are natural generalizations of Longest Path. Our approach uses bidimensionality theory combined with novel dynamic programming techniques over branch decompositions of the input graph. These techniques can be applied to a more general family of problems that deal with finding connected subgraphs under certain degree constraints.

Keywords: parameterized complexity, planar graphs, subexponential algorithm, branch decomposition, graph minors, bidimensionality, Catalan structures.

7.1 Introduction

During the last years a considerable amount of work has been devoted to design subexponential parameterized algorithms for NP-hard optimization problems on planar graphs and, more generally, on sparse classes of graphs [93, 96–100, 140]. In this chapter we apply the general approach of [93, 96–100, 140] to a family of problems dealing with finding connected subgraphs under degree constraints. Along the way, we introduce novel dynamic programming techniques over branch decompositions that can be applied to more general classes of problems.

We define the following family of problems for $d \geq 2$.

Maximum d-Degree-Bounded Connected Subgraph (MDBCS$_d$)
Input: A graph G and a non-negative integer k.
Question: Does G contain a connected subgraph H with maximum degree at most d and at least k edges?

If $d = 2$ the problem is equivalent to the Longest Path (or Cycle, if G is Hamiltonian) problem, hence MDBCS$_d$ is a generalisation of it. MDBCS$_d$ is one of the classical NP-hard problems listed in [132], and we have seen in Chapter 5 that it is not in Apx for any $d \geq 2$. Without the connectivity constraint, the problem is known to be in P using matching techniques [158]. When the problem is parameterized by k we denote it by k-MDBCS$_d$. (We refer to [121] for an introduction to parameterized complexity.) Our target is to find $2^{O(\sqrt{k})} \cdot O(n)$ step algorithms to solve it when the input is restricted to planar graphs. Section 7.3 is devoted to obtain combinatorial bounds using bidimensionality theory. Section 7.4 presents new dynamic programming techniques, that can be applied to general graphs. In Section 7.5 we see how to speed-up these algorithms when the input is restricted to planar graphs, using Catalan structures. This strategy can be extended to several related problems asking for a maximum connected subgraph satisfying certain degree constraints, as discussed in Section 7.6. Some open problems are listed in Section 7.7. We first provide some background in Section 7.2.

7.2 Background

Recall the definition of branchwidth from Section I.1.1 (page 16) and the definition of graph minor from Section I.1.2 (page 16). Recall also that a parameter **P** defined on simple undirected graphs is *closed under taking of minors* (or simply *minor closed*) if $G' \preceq G \Rightarrow \mathbf{P}(G') \leq \mathbf{P}(G)$ (here "$\preceq$" denotes the minor relation). The following fundamental theorem states that square grids are the obstruction for branchwidth on planar graphs.

Theorem 7.1 (Robertson, Seymour, and Thomas [175]) *Let $\ell \geq 1$ be an integer. Every planar graph of branchwidth at least ℓ contains an $(\lfloor \ell/4 \rfloor \times \lfloor \ell/4 \rfloor)$-grid as a minor.*

A parameter **P** is *minor bidimensional* [92] with *density* δ if

- P is minor closed, and
- for the $(r \times r)$-grid R, $P(R) = (\delta r)^2 + o((\delta r)^2)$.

Theorem 7.1 implies the following useful property.

Lemma 7.1 (Demaine et al. [92]) *If* **P** *is a bidimensional parameter with density δ then for any planar graph G, $\mathbf{bw}(G) \leq \frac{4}{\delta} \cdot \sqrt{\mathbf{P}(G)} + O(1)$.*

There is a recent and powerful theory about bidimensional parameters, called *bidimensionality theory*, that has proved very useful in the design of subexponential exact and parameterized algorithms for many hard problems [93, 96–100, 140].

7.3 Bounds for Branchwidth

We define the following parameter on simple undirected graphs.

$$\textbf{medbcs}_d(G) = \max\{|E(H)| \mid H \subseteq G \ \wedge \ H \text{ is connected } \wedge \ \Delta(H) \leq d\}.$$

Lemma 7.2 *For any integer $d \geq 1$, the parameter* $\textbf{medbcs}_d$ *is minor closed.*

Proof: If G' occurs from G after an edge removal, then clearly $\textbf{medbcs}_d(G') \leq \textbf{medbcs}_d(G)$. Let us see that the same holds if G' occurs from G after the contraction of an edge $\{x, y\}$. Indeed, we shall see that given any connected subgraph $H' \subseteq G'$ with $\Delta(H') \leq d$, we can find a connected subgraph $H^* \subseteq G$ with $\Delta(H^*) \leq d$ and $|E(H^*)| \geq |E(H')|$. Let H be the major of H' in G. We can assume that $v_{xy} \in V(H')$, otherwise we set $H^* = H$. We define $N_{xy} = N_H(x) \cap N_H(y)$, $N_{x-y} = N_H(x) - N_{xy} - \{y\}$, and $N_{y-x} = N_H(y) - N_{xy} - \{x\}$. The subgraph H is connected and $|E(H)| \geq |E(H')|$, but the vertices x, y, and those in N_{xy} may have degree $d+1$. Since $\Delta(H') \leq d$, also $|N_{H'}(v_{xy})| = |N_{x-y}| + |N_{y-x}| + |N_{xy}| \leq d$. Suppose w.l.o.g. that $|N_{x-y}| \geq |N_{y-x}|$. We distinguish several cases to define the subgraph H^*: If $|N_{x-y}| = d$, let $H^* = (V(H) - \{y\}, E(H) - \{x, y\})$. Suppose henceforth that $|N_{x-y}| < d$. If $|N_{xy}| = 0$, let $H^* = H$. If $N_{xy} = \{z_1\}$, let $H^* = (V(H), E(H) - \{x, z_1\})$. Finally, if $N_{xy} = \{z_1, \ldots, z_k\}$ for some $k \geq 2$, let $H^* = (V(H), E(H) - \{x, z_1\} - \cup_{i=2}^{k}\{y, z_i\})$. It is easy to check that, in all cases, the subgraph H^* is connected, $\Delta(H^*) \leq d$, and $|E(H^*)| \geq |E(H')|$. □

Using Lemmas 7.2 and 7.1 we can obtain a combinatorial bound of the parameter $\textbf{medbcs}_d$ in terms of the branchwidth of the planar graph G.

Lemma 7.3 *For any $d \geq 2$ and for any planar graph G it holds that*

$$\textbf{bw}(G) \leq \frac{4}{\delta} \cdot \sqrt{\textbf{medbcs}_d(G)} + O(1), \quad \textit{with} \quad \delta = \begin{cases} 1 & \text{, if } d = 2 \\ \sqrt{3/2} & \text{, if } d = 3 \\ \sqrt{2} & \text{, if } d \geq 4 \end{cases}$$

Proof: We shall prove that the parameter $\textbf{medbcs}_d(G)$ is bidimensional for any $d \geq 2$. It is minor closed due to Lemma 7.2. Let us see how the parameter behaves on the grid. Let R be an $(r \times r)$-grid. If $d = 2$, then clearly $\textbf{medbcs}_2(R) \geq r^2 - 1$ (or r^2 if r is even, because in this case the grid contains a Hamiltonian cycle). That is, the density of $\textbf{medbcs}_2$ is 1. If $d \geq 4$ then the optimal solution contains all the edges, i.e., $\textbf{medbcs}_d(R) = 2r(r-1)$. Said otherwise, the density is $\sqrt{2}$. Finally, if $d = 3$, we shall see that $\textbf{medbcs}_3(R) \geq 2r(r-1) - \left\lceil \frac{r-2}{2} \right\rceil (r-2)$. Such a solution is obtained in the following way. Take all the *horizontal* edges of the grid, and the *vertical* edges corresponding to the first and the last column. Then, beginning from the first row, take alternatively the remaining vertical edges (see Figure 7.1 for an illustration). One can easily check that the subgraph obtained in this way is connected, has maximum degree 3 and has $2r(r-1) - \left\lceil \frac{r-2}{2} \right\rceil (r-2)$ edges. That is, the density of $\textbf{medbcs}_3$ is at least $\sqrt{3/2}$. The result follows from Lemma 7.1. □

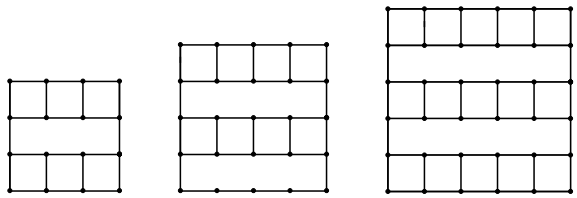

Figure 7.1: Connected subgraphs with maximum degree 3 on (4×4), (5×5), and (6×6)-grids respectively, used in the proof of Lemma 7.3.

7.4 The Algorithms

Let G be in this section a (not necessarily planar) graph on n vertices. We denote the *empty set* by $\emptyset$ and the *empty function* by $\varnothing$. Let (T, μ) be a branch decomposition of width $\leq \ell$ of G. We pick an arbitrary edge $e^* \in E(T)$, we subdivide it by adding a new vertex v_{new} and then add a new vertex r and make it adjacent to v_{new}. We extend μ by setting $\mu(r) = \emptyset$ and we root T at vertex r. For each $e \in E(T)$ let T_e be the tree of the forest $T \backslash e$ that does not contain r as a leaf (i.e., the tree that is "below" e in the rooted tree T) and let E_e be the edges that are images, via μ, of the leaves of T that are also leaves of T_e. We denote $G_e = G[E_e]$. Observe that, if $e_r = \{v_{\text{new}}, r\}$, then $G_{e_r} = G$.

Given a set A, we define a *d-weighted packing* of A as any pair $(\mathcal{A}, \psi)$ where $\mathcal{A}$ is a (possible empty) collection of mutually disjoint nonempty subsets of A and $\psi : A \to \{0, \ldots, d\}$ is a mapping corresponding integers from 0 to d to the elements of A. It will be convenient to think of such a packing $\mathcal{A}$ of A as a hypergraph $\mathcal{G} = (A, \mathcal{A})$. Note that, by definition, $\mathcal{A}$ is a matching in $\mathcal{G}$. For convenience, given such a collection $\mathcal{A}$, we denote by $\cup\mathcal{A}$ the set $\bigcup_{X \in \mathcal{A}} X$.

Let $(\mathcal{A}, \psi)$ and $(\mathcal{A}', \psi')$ be two d-weighted packings of two sets A and A'. We define $(\mathcal{A}, \psi) \oplus (\mathcal{A}', \psi')$ as the $2d$-weighted packing $(\mathcal{A}'', \psi'')$ of $A'' = A \cup A'$ where $\mathcal{A}''$ is the packing of A'' defined by the connected components of the hypergraph $(A \cup A', \mathcal{A} \cup \mathcal{A}')$ (i.e., the nonempty subsets of the packing $\mathcal{A}''$ are the vertex sets corresponding to the connected components of the hypergraph $(A \cup A', \mathcal{A} \cup \mathcal{A}')$) and where for any $x \in A \cup A'$,

$$\psi''(x) = \begin{cases} \psi(x) & \text{, if } x \in A - A' \\ \psi(x) + \psi'(x) & \text{, if } x \in A \cap A' \\ \psi'(x) & \text{, if } x \in A' - A \end{cases}$$

If $(\mathcal{A}, \psi)$ is a d-weighted packing of a set A and $A' \subseteq A$, we define $(\mathcal{A}, \psi)|_{A'}$ as the d-weighted packing $(\mathcal{A}', \psi')$ of the set A' where $\mathcal{A}' = \{X \cap A' \mid X \in \mathcal{A}\}$ and $\psi' = \{(x, \psi(x)) \mid x \in A'\}$.

Let $\mathscr{P}_e$ be the collection of all d-weighted packings $(\mathcal{A}, \psi)$ of $\mathbf{mid}(e)$, and let $\ell = |\mathbf{mid}(e)|$. Observe that if $e_r = \{v_{\text{new}}, r\}$, then $\mathscr{P}_{e_r} = \{(\emptyset, \varnothing)\}$. We use the notation $C(H)$ for the set of

connected components of a graph (or hypergraph) H. Given $(\mathcal{A}, \psi) \in \mathscr{P}_e$ we define

$$\begin{aligned}\mathbf{opt}_e(\mathcal{A},\psi) \;=\; \max\{\{0\} \cup \{|E(H)| \;:\; & \exists\, H \subseteq G_e : \Delta(H) \leq d \;\wedge \\ & \text{if } (\mathcal{A} \neq \emptyset) \text{ then} \\ & \quad \{V(H') \cap \mathbf{mid}(e) \mid H' \in \mathcal{C}(H)\} = \mathcal{A} \;\wedge \\ & \quad \{(v, \mathbf{deg}_H(v)) \mid v \in \cup_{A\in\mathcal{A}} A\} = \psi \\ & \text{else if } (\mathcal{A} = \emptyset) \text{ then} \\ & \quad |\mathcal{C}(H)| \leq 1 \;\wedge\; V(H) \cap \mathbf{mid}(e) = \emptyset \;\}\}\end{aligned}$$

Clearly, $\mathbf{opt}_{e_r}(\emptyset, \varnothing) = \mathbf{medbcs}_d(G)$. The idea is the following:

- If $\mathcal{A} \neq \emptyset$, we look for the best solution H in the graph G_e such that its restriction to $\mathbf{mid}(e)$ induces the connected components given by $\mathcal{A}$ and obeys the degrees given by ψ.

- Otherwise, if $\mathcal{A} = \emptyset$, we look for the best solution H in G_e not intersecting $\mathbf{mid}(e)$. Since $\mathbf{mid}(e)$ is a separator of G, it is clear that in this case the solution H must be a connected subgraph of G_e disjoint from $\mathbf{mid}(e)$.

Let us now see how these values of $\mathbf{opt}_e(\mathcal{A}, \psi)$ can be explicitly computed using dynamic programming over a branch decomposition of G.

Let e, e_1, e_2 be three edges of T that are incident to the same vertex and such that e is closer to the root of T than the other two (see the upper part of Figure 7.2). To perform the *join/forget* operations in the middle set $\mathbf{mid}(e)$, we distinguish two cases according to the packing $\mathcal{A}$ of $\mathbf{mid}(e)$:

(1) In the case $\mathcal{A} \neq \emptyset$, the value of $\mathbf{opt}_e(\mathcal{A}, \psi)$ is given by

$$\begin{aligned}\mathbf{opt}_e(\mathcal{A},\psi) \;=\; \max\{\{0\} \cup \{l : \exists (\mathcal{A}_i, \psi_i) \in \mathscr{P}_{e_i}, i = 1,2, \text{ such that} \\ \cup\, \mathcal{A}_1 \cap (\mathbf{mid}(e_1) \cap \mathbf{mid}(e_2)) = \cup \mathcal{A}_2 \cap (\mathbf{mid}(e_1) \cap \mathbf{mid}(e_2)) \wedge \\ (\mathcal{A}_1, \psi_1) \oplus (\mathcal{A}_2, \psi_2) \text{ is a } d\text{-weighted} \\ \quad \text{packing of } \mathbf{mid}(e_1) \cup \mathbf{mid}(e_2) \;\wedge \\ (\mathcal{A}, \psi) = ((\mathcal{A}_1, \psi_1) \oplus (\mathcal{A}_2, \psi_2))|_{\mathbf{mid}(e)} \;\wedge \\ \text{if } (\mathcal{A}_1 = \emptyset) \text{ then } l = \mathbf{opt}_{e_2}(\mathcal{A}_2, \psi_2) \\ \text{if } (\mathcal{A}_2 = \emptyset) \text{ then } l = \mathbf{opt}_{e_1}(\mathcal{A}_1, \psi_1) \\ \text{else } l = \mathbf{opt}_{e_1}(\mathcal{A}_1, \psi_1) + \mathbf{opt}_{e_2}(\mathcal{A}_2, \psi_2) \;\}\}\end{aligned}$$

(2) In the case $\mathcal{A} = \emptyset$, the value of $\mathbf{opt}_e(\emptyset, \psi)$ is given by

$$\begin{aligned}
\mathbf{opt}_e(\emptyset,\psi) \;=\; \max\{\{0\} \cup \{l : \exists (\mathcal{A}_i,\psi_i) \in \mathscr{P}_{e_i}, i=1,2, \text{ such that} \\
\cup \mathcal{A}_1 \cap (\mathbf{mid}(e_1) \cap \mathbf{mid}(e_2)) = \cup \mathcal{A}_2 \cap (\mathbf{mid}(e_1) \cap \mathbf{mid}(e_2)) \wedge \\
(\mathcal{A}_1,\psi_1) \oplus (\mathcal{A}_2,\psi_2) \text{ is a } d\text{-weighted} \\
\quad \text{packing of } \mathbf{mid}(e_1) \cup \mathbf{mid}(e_2) \;\wedge \\
(\emptyset,\psi) = ((\mathcal{A}_1,\psi_1) \oplus (\mathcal{A}_2,\psi_2))|_{\mathbf{mid}(e)} \;\wedge \\
\text{if } (\mathcal{A}_1 = \emptyset \;\wedge\; \mathcal{A}_2 = \emptyset) \text{ then} \\
\quad l = \max\{\mathbf{opt}_{e_1}(\mathcal{A}_1,\psi_1), \mathbf{opt}_{e_2}(\mathcal{A}_2,\psi_2)\} \\
\text{if } (\mathcal{A}_1 \neq \emptyset \;\wedge\; \mathcal{A}_2 = \emptyset) \text{ then} \\
\quad l = \max\{\mathbf{opt}_{e_2}(\mathcal{A}_2,\psi_2), \{\mathbf{opt}_{e_1}(\mathcal{A}_1,\psi_1)|_X : X \in \mathcal{A}_1\}\} \\
\text{if } (\mathcal{A}_1 = \emptyset \;\wedge\; \mathcal{A}_2 \neq \emptyset) \text{ then} \\
\quad l = \max\{\mathbf{opt}_{e_1}(\mathcal{A}_1,\psi_1), \{\mathbf{opt}_{e_2}(\mathcal{A}_2,\psi_2)|_X : X \in \mathcal{A}_2\}\} \\
\text{if } (\mathcal{A}_1 \neq \emptyset \;\wedge\; \mathcal{A}_2 \neq \emptyset) \text{ then} \\
\quad l = \max\{\mathbf{opt}_{e_1}(X,\psi_1)|_{\mathbf{mid}(e_1)} + \mathbf{opt}_{e_2}(X,\psi_2)|_{\mathbf{mid}(e_2)} : \\
\qquad X \in C(\mathbf{mid}(e_1) \cup \mathbf{mid}(e_2), \mathcal{A}_1 \cup \mathcal{A}_2)\}\}
\end{aligned}$$

These ideas are schematically illustrated in Figure 7.2. Finally, suppose that $e_{\text{leaf}} = \{x, y\} \in E(T)$ is an edge such that either x or y is a leaf of T. Let $\{v_1, v_2\} \in E(G)$ be the image under μ of the endpoint of e which is a leaf of T. Then

$$\mathbf{opt}_{e_{\text{leaf}}}(\mathcal{A},\psi) = \begin{cases} 1 & \text{, if } (\mathcal{A} = \{\{v_1, v_2\}\} \quad \wedge \quad \psi = \{(v_1,1),(v_2,1)\}) \\ 0 & \text{, otherwise} \end{cases}$$

Running time. The size of the tables of the dynamic programming over the branch decomposition of the input graph, namely $|\mathscr{P}_e|$, determines the running time of our algorithms. The number of ways a set of ℓ elements can be partitioned into nonempty subsets is well-known as the ℓ-th *Bell number* [115] and is denoted by B_ℓ. We can express $|\mathscr{P}_e|$ in terms of the Bell numbers:

$$|\mathscr{P}_e| \;=\; (d+1)^\ell \cdot \sum_{i=0}^{\ell} \binom{\ell}{i} B_{\ell-i} \;\leq\; (d+1)^\ell \cdot 2^{2\ell \cdot \log \ell}, \tag{7.1}$$

where the last inequality is an easy exercise using that $B_\ell \leq \frac{e^\ell - 1}{(\log \ell)^\ell} \ell!$ [115]. At each edge e of the branch decomposition, to compute all the values $\mathbf{opt}_e(\mathcal{A},\psi)$ we test all the possibilities of combining d-weighted packings of the two middle sets $\mathbf{mid}(e_1)$ and $\mathbf{mid}(e_2)$. The operations $(\mathcal{A}_1,\psi_1) \oplus (\mathcal{A}_2,\psi_2)$ and $(\mathcal{A},\psi)|_{A'}$ take $O(|\mathbf{mid}(e)|)$ time. Let $m = |E(G)|$. Hence, by Equation (7.1), given a branch decomposition of a general graph G of width at most ℓ, the value of $\mathbf{medbcs}_d(G)$ can be computed in $(d+1)^{2\ell} \cdot 2^{4\ell \cdot \log \ell} \cdot \ell \cdot m$ steps.

7.5 Speed-up for Planar Graphs using Catalan Structures

In this section we will see that when the input is restricted to planar graphs the term $2^{O(\ell \cdot \log \ell)}$ in Equation (7.1) can be reduced to $2^{O(\ell)}$. We need first some definitions.

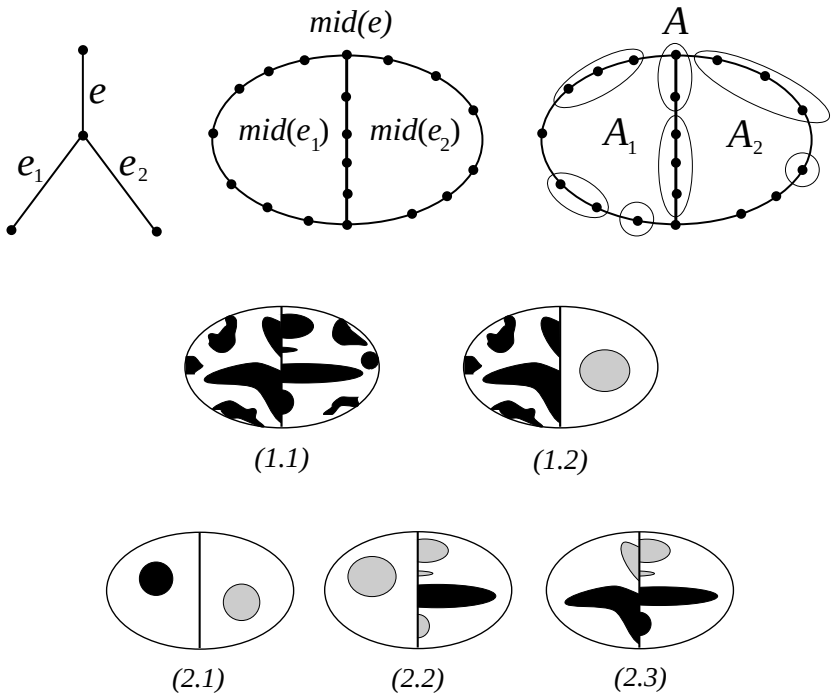

Figure 7.2: *Join/forget* operations in the dynamic programming over a branch decomposition. The dark regions represent an optimal subgraph in each case. Case *(1)*: $\mathcal{A} \neq \emptyset$; *(1.1)* $\mathcal{A}_1 \neq \emptyset, \mathcal{A}_2 \neq \emptyset$; *(1.2)* $\mathcal{A}_1 \neq \emptyset, \mathcal{A}_2 = \emptyset$. Case *(2)*: $\mathcal{A} = \emptyset$; *(2.1)* $\mathcal{A}_1 = \emptyset, \mathcal{A}_2 = \emptyset$; *(2.2)* $\mathcal{A}_1 = \emptyset, \mathcal{A}_2 \neq \emptyset$; *(2.3)* $\mathcal{A}_1 \neq \emptyset, \mathcal{A}_2 \neq \emptyset$.

Let G be a planar graph embedded on a sphere $\mathbb{S}$. An *O-arc* is a subset of $\mathbb{S}$ homeomorphic to a circle. An O-arc in S is called a *noose* of the embedding of G if it meets G only in vertices. A *sphere cut decomposition* or *sc-decomposition* (T, μ, π) of G is a branch decomposition of G with the following property: for every edge e of T, there exists a noose O_e meeting every face at most once and bounding the two open discs Δ_1 and Δ_2 such that $G_i \subseteq \Delta_i \cup O_e$, $1 \leq i \leq 2$. Thus O_e meets G only in $\mathbf{mid}(e)$ and its length is $|\mathbf{mid}(e)|$. A *clockwise traversal* of O_e in the embedding of G defines the cyclic ordering π of $\mathbf{mid}(e)$. We always assume that the vertices of every middle set $\mathbf{mid}(e) = V(G_1) \cap V(G_2)$ are enumerated according to π.

Theorem 7.2 (Seymour and Thomas [180]) *Let G be a planar graph of branchwidth at most ℓ without vertices of degree one embedded on a sphere. Then there exists an sc-decomposition of G of width at most ℓ.*

In addition, such an sc-decomposition can be constructed in time $O(n^3)$ [139].

The size of the tables of the dynamic programming algorithm is given by in how many ways a solution of k-MDBCS$_d$ in G_e can intersect $\mathbf{mid}(e)$. Let (T, μ, π) be a sphere cut decomposition of width $\leq \ell$, and we can assume $\ell \leq \mathbf{bw}(G)$ by Theorem 7.2. Then the

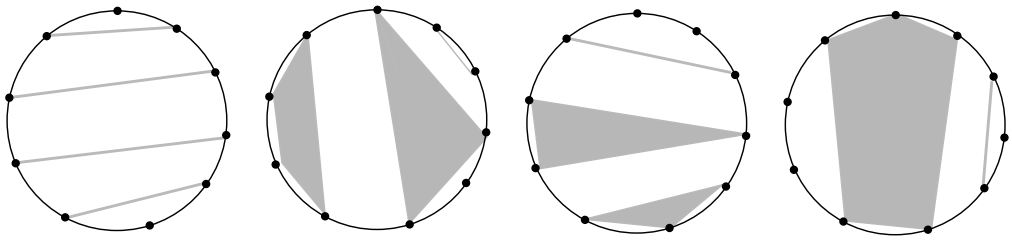

Figure 7.3: Catalan structures in the middle set of a sphere cut decomposition.

vertices of $\mathbf{mid}(e)$ are situated around a noose. A *non-crossing partition (ncp)* is a partition $P(n) = \{P_1, \ldots, P_m\}$ of the set $S = \{1, \ldots, n\}$ such that there are no numbers $a < b < c < d$ where $a, c \in P_i$, and $b, d \in P_j$ with $i \neq j$.

When we restrict the input graph G to be planar, then the subgraph given by the intersection of a partial solution of k-MDBCS$_d$ in G_e with $\mathbf{mid}(e)$ is also planar. The reduction from $2^{O(\ell \cdot \log \ell)}$ to $2^{O(\ell)}$ is based on calculating in how many ways we can draw hyperedges inside a cycle such that they touch the cycle on its vertices and they do not share common internal points in the plain (they do not intersect), as it is illustrated in Figure 7.3.

The number of such configurations is closely related to the number of *non-crossing partitions* over ℓ vertices, which is equal to the ℓ-th *Catalan number* $\mathrm{CN}(\ell) = \frac{1}{\ell+1}\binom{2\ell}{\ell} \sim \frac{4^\ell}{\sqrt{\pi}\ell^{3/2}} \leq 4^\ell$ [155].

Indeed, in the same spirit of Equation (7.1) we can write

$$\begin{aligned} |\mathscr{P}_e| &= (d+1)^\ell \cdot \sum_{i=0}^{\ell} \binom{\ell}{i} \mathrm{CN}(\ell - i) \leq (d+1)^\ell \cdot \sum_{i=0}^{\ell} \binom{\ell}{i} 4^{\ell-i} = \\ &= (d+1)^\ell 4^\ell \cdot \sum_{i=0}^{\ell} \binom{\ell}{i} \left(\frac{1}{4}\right)^i = (d+1)^\ell 4^\ell \cdot \left(1 + \frac{1}{4}\right)^\ell = (d+1)^\ell \cdot 5^\ell. \end{aligned}$$

Since G is planar, $|E(G)| = O(|V(G)|)$, hence so is the number of middle sets in any branch decomposition of G. Therefore,

Proposition 7.1 *For every planar graph G and given a sphere cut decomposition (T, μ, π) of G of width $\leq \ell$, the value of $\mathbf{medbcs}_d(G)$ can be computed in $O\left((d+1)^{2\ell} \cdot 5^{2\ell} \cdot \ell \cdot n\right)$ steps.*

Let δ be the constant defined in Lemma 7.3. Summarizing,

Theorem 7.3 k-Planar Maximum d-Degree-Bounded Connected Subgraph *is solvable in time $O\left(2^{\log(5(d+1))8\sqrt{k}/\delta}\sqrt{k} \cdot n + n^3\right)$ for any $d \geq 2$.*

Proof: First, using Theorem 7.2, we construct in time $O(n^3)$ an optimal sphere cut decomposition of G of width $\mathbf{bw}(G)$. We distinguish two cases: If $\mathbf{bw}(G) > 4/\delta \cdot \sqrt{k}$, then by Lemma 7.3 the answer to the parameterized problem is automatically YES. Otherwise, $\mathbf{bw}(G) \leq 4/\delta \cdot \sqrt{k}$ and the value of $\mathbf{medbcs}_d(G)$ can be computed by Proposition 7.1 in time $O\left((d+1)^{8\sqrt{k}/\delta} \cdot 5^{8\sqrt{k}/\delta} \cdot 4/\delta\sqrt{k} \cdot n\right) = O\left(2^{\log(5(d+1))8\sqrt{k}/\delta}\sqrt{k} \cdot n\right)$. □

7.6 Extensions

Appropriate modifications of the dynamic programming algorithm of Section 7.4 allow us to obtain also subexponential parameterized algorithms for the variant of the problem in which the aim is to maximize the number of vertices of the subgraph H, as well as for the variant in which the output subgraph is required to be induced (for both the edge and vertex maximization versions). Another variant is when the list of prescribed degrees of the vertices belongs to a subset of $\mathbb{Z}_q$ for a fixed integer q. Finally, we discuss how to transform these parameterized algorithms into subexponential *exact* algorithms on planar graphs.

7.6.1 Maximizing the number of vertices

In this section we focus on the following family of problems for $d \geq 2$:

Vertex Maximum d-Degree-Bounded Connected Subgraph (VMDBCS$_d$)
Input: A graph G and a non-negative integer k.
Question: Does G contain a connected subgraph H with $\Delta(H) \leq d$ and $\lvert V(H)\rvert \geq k$?

In order to obtain subexponential parameterized algorithms for VMDBCS$_d$ on planar graphs, let us see how the techniques presented in the preceding sections must be modified. The corresponding parameter is

$$\mathbf{mvdbcs}_d(G) = \max\{|V(H)| \mid H \subseteq G \wedge H \text{ is connected} \wedge \Delta(H) \leq d\}.$$

First, it is easy to check that Lemmas 7.2 and 7.3 hold for the parameter $\mathbf{mvdbcs}_d(G)$ with $\delta = 1$ for any $d \geq 2$. Secondly, the dynamic programming approach of Section 7.4 remains the same, except for the following modifications.

When computing a partial solution $\mathbf{opt}_e(\mathcal{A}, \psi)$ in G_e from the partial solutions in G_{e_1} and G_{e_2}, we have to be careful in order to avoid counting twice the vertices that belong to both $\mathbf{mid}(e_1)$ and $\mathbf{mid}(e_2)$. More precisely,

- in the case $\mathcal{A} \neq \emptyset$, $\mathcal{A}_1 \neq \emptyset$, and $\mathcal{A}_2 \neq \emptyset$ we have that

$$l = \mathbf{opt}_{e_1}(\mathcal{A}_1, \psi_1) + \mathbf{opt}_{e_2}(\mathcal{A}_2, \psi_2) - |V(\mathcal{G}[\mathcal{A}_1]) \cap V(\mathcal{G}[\mathcal{A}_2])|,$$

  where $\mathcal{G}[\mathcal{A}_i]$, $i = 1, 2$, denotes the hypergraph induced by the hyperedges in $\mathcal{A}_i$; and

- in the case $\mathcal{A} = \emptyset$, $\mathcal{A}_1 \neq \emptyset$, and $\mathcal{A}_2 \neq \emptyset$ we have that

$$\begin{aligned} l \;=\; \max\{&\mathbf{opt}_{e_1}(X, \psi_1)|_{\mathbf{mid}(e_1)} + \mathbf{opt}_{e_2}(X, \psi_2)|_{\mathbf{mid}(e_2)} - \\ &|V(\mathcal{G}[X]) \cap \mathbf{mid}(e_1) \cap \mathbf{mid}(e_2)| \;: \\ &X \in \mathcal{C}(\mathbf{mid}(e_1) \cup \mathbf{mid}(e_2), \mathcal{A}_1 \cup \mathcal{A}_2)\}\;\}. \end{aligned}$$

Also, if $e_{\text{leaf}} = \{x, y\} \in E(T)$ is an edge such that x is a leaf of T, and $\{v_1, v_2\} \in E(G)$ is the image of x under μ, then

$$\mathbf{opt}_{e_{\text{leaf}}}(\mathcal{A}, \psi) = \begin{cases} 2 & \text{, if } (\mathcal{A} = \{\{v_1, v_2\}\} \quad \wedge \quad \psi = \{(v_1, 1), (v_2, 1)\}) \\ & \quad \vee (\mathcal{A} = \{\{v_1\}, \{v_2\}\} \quad \wedge \quad \psi = \{(v_1, 0), (v_2, 0)\}) \\ 1 & \text{, if } (\mathcal{A} = \{\{v_1\}\} \quad \wedge \quad \psi = \{(v_1, 0), (v_2, 0)\}) \\ & \quad \vee (\mathcal{A} = \{\{v_2\}\} \quad \wedge \quad \psi = \{(v_1, 0), (v_2, 0)\}) \\ 0 & \text{, otherwise} \end{cases}$$

Finally, the speed-up described in Section 7.5 can be applied directly to VMDBCS$_d$, since Catalan structures also appear in the middle sets of a sc-decomposition of the planar input graph. Summarizing,

Theorem 7.4 *k-Planar Vertex Maximum d-Degree-Bounded Connected Subgraph is solvable in time $O\left(2^{\log(5(d+1))8\sqrt{k}}\sqrt{k} \cdot n + n^3\right)$ for any $d \geq 2$.*

7.6.2 Looking for an induced subgraph

It is also natural to ask, instead of for a subgraph H of the input graph G, for an *induced* subgraph H. In this section we focus on the edge-maximization version of the problem, the modifications for the node-maximization version being analogous to those described in Section 7.6.1. We denote the problem by Maximum d-Degree-Bounded Connected Induced Subgraph (MDBCIS$_d$).

In contrast to the dynamic programming presented in Section 7.4, now we need only to consider those packings $\mathcal{A}$ of $\mathbf{mid}(e)$ that "respect" the fact that the solution subgraph must be induced. For this, the only difference from the algorithms for the non-induced case is that the optimal solution in the leaves of the branch decomposition becomes

$$\mathbf{opt}_{e_{\text{leaf}}}(\mathcal{A}, \psi) = \begin{cases} 1 & \text{, if } (\mathcal{A} = \{\{v_1, v_2\}\} \quad \wedge \quad \psi = \{(v_1, 1), (v_2, 1)\} \\ 0 & \text{, otherwise} \end{cases}$$

Since for any middle set $\mathbf{mid}(e)$, $\mathscr{P}_e^{\text{ind}} \subseteq \mathscr{P}_e$, the running time of Theorem 7.3 also applies to k-Planar Maximum d-Degree-Bounded Connected Induced Subgraph, replacing the constant δ with $\delta/\sqrt{2}$ when $d \in \{2, 3\}$, due to the fact that the optimal subgraphs of MDBCIS$_d$ on the square grid (see Lemma 7.2) must be induced.

7.6.3 More general constraints on the degree

All the variants of the problem considered so far have in common that the degree of any vertex belonging to the output subgraph must lie in the interval $[0, d]$. It makes sense to consider a more general version in which the interval of allowed degrees depends on each vertex. Namely, for each vertex $v \in V(G)$ we are given an interval $I_v = [\ell_v, r_v]$ and we look for a maximum connected subgraph H in which the degree of each vertex v lies in I_v. (If $0 \in I_v$ then vertex v may not belong to $V(H)$.) When the output subgraph is not required to be connected, some variants of the problem are in P and some others

become NP-hard [158]. In general, we cannot guarantee that the parameters associated with this general problem are minor closed, hence the approach used with MDBCS$_d$ does not carry over. Nevertheless, we can obtain an algorithm to solve it similar to the one of Proposition 7.1, replacing the term $(d+1)^{2\ell}$ with $(\max_{v\in V(G)} r_v + 1)^{2\ell}$. The ideas behind the dynamic programming are essentially the same.

Another variant is obtained when forcing the allowed degrees to belong to a subset of $\mathbb{Z}_q$ for some fixed integer q. In this case it is not difficult to see that the term $(d+1)^{2\ell}$ can be replaced with $q^{2\ell}$. For instance, the case where all the degrees are required to be 0 (mod 2) corresponds to the Maximum Eulerian Subgraph problem. This approach, given a planar graph with a sphere cut decomposition of width $\leq \ell$, yields an algorithm to solve Maximum Eulerian Subgraph in time $O\left(2^{2\ell} \cdot 5^{2\ell} \cdot \ell \cdot n\right)$.

7.6.4 Exact algorithms

The subexponential parameterized algorithms we have presented on planar graphs can be naturally transformed to subexponential *exact* algorithms by using that for any planar graph G, $\mathbf{bw}(G) \leq \sqrt{4.5 \cdot |V(G)|}$ [123].

Indeed, given a planar graph G and a sphere cut decomposition of width $\leq \sqrt{4.5 \cdot |V(G)|}$, we can compute an optimal solution of MDBCS$_d$ in G in $O\left((d+1)^{4.24\sqrt{n}} \cdot 5^{4.24\sqrt{n}} \cdot n^{3/2}\right)$ steps (by Proposition 7.1). The same argument applies to all the variants of the problem discussed above.

In addition, we can derive a subexponential exact algorithm for the following problem on planar graphs: Minimum Degree Spanning Tree (MDST). The MDST problem consists in, given an undirected unweighted graph G, finding a spanning tree of G with minimizes the maximum degree over all the spanning trees of G. This problem has been widely studied in the literature (cf. for instance [128]), and we are unaware of the existence of subexponential exact algorithms on planar graphs. Our algorithm works as follows: given a planar graph G, we find an optimal solution H_d of VMDBCS$_d$ in G for $d = 2, \ldots, n-1$. Let d^* be the first value of d for which $|V(H_d)| = n$. Then an optimal solution of MDST in G is given by any spanning tree of H_{d^*}.

A graph is *supereulerian* if it has a spanning Eulerian subgraph [66]. Combining the ideas of the algorithm above with the ideas of Section 7.6.3 yields a subexponential exact algorithm to decide whether a planar graph is supereulerian or not.

7.7 Conclusions

In this chapter we obtained a $2^{O(\sqrt{k})}n^{O(1)}$ algorithm for k-MDBCS$_d$ and related problems on planar graphs. Several interesting problems remain open. First, it seems natural to try to improve the worst-case running time of our algorithms. Much more challenging is to find subexponential parameterized algorithms for the edge- or node-weighted versions of the problem. Actually, the weighted versions of our parameters remain minor closed (by an easy modification of Lemma 7.2), however the fundamental difference is that the

combinatorial bound of Lemma 7.3 does not hold anymore. On the other hand, the natural extension of this chapter would be to conceive subexponential parameterized algorithms for k-MDBCS$_d$ on other sparse graph classes, like graphs of bounded genus and, more generally, minor-free families of graphs.

Finally, note that the MDBCS$_d$ problem is equivalent to finding a maximum connected subgraph not containing the star $K_{1,d+1}$ as a topological minor. Many classical NP-hard problems can be expressed as finding a maximum subgraph excluding a fixed graph H as a minor (or induced minor, or subgraph, or induced subgraph, or topological minor), hence conceiving a general framework to design subexponential parameterized algorithms for this class of problems would be a celebrated result.

Chapter 8

Dynamic Programming for Graphs on Surfaces

In this chapter we provide a framework for the design of $2^{\mathcal{O}(k)} \cdot n$ step dynamic programming algorithms for surface-embedded graphs on n vertices of branchwidth at most k. Our technique applies to graph problems for which dynamic programming uses tables encoding set partitions. For general graphs, the best known algorithms for such problems run in $2^{\mathcal{O}(k \cdot \log k)} \cdot n$ steps. That way, we considerably extend the class of problems that can be solved by algorithms whose running times have a *single exponential dependence* on branchwidth, and improve the running time of several existing algorithms. Our approach is based on a new type of branch decomposition called *surface cut decomposition*, which generalizes sphere cut decompositions for planar graphs (see page 168), and where dynamic programming should be applied for each particular problem. The construction of such a decomposition uses a new graph-topological tool called *polyhedral decomposition*. The main result is that if dynamic programming is applied on surface cut decompositions, then the time dependence on branchwidth is *single exponential*. This fact is proved by a detailed analysis of how non-crossing partitions are arranged on surfaces with boundary and uses diverse techniques from topological graph theory and analytic combinatorics.

Keywords: parameterized algorithms, analytic combinatorics, graphs on surfaces, branchwidth, dynamic programming, polyhedral embeddings, symbolic method, non-crossing partitions.

8.1 Introduction

One of the most important parameters in the design and analysis of graph algorithms is the branchwidth of a graph. Branchwidth, together with its twin parameter of treewidth, can be seen as a measure of the topological resemblance of a graph to a tree. Its algorithmic importance has its origins in the celebrated theorem of Courcelle (see e.g. [84]), stating

that graph problems expressible in monadic second-order logic can be solved in $f(\mathbf{bw}) \cdot n$ (here **bw** is the branchwidth[1] and n is the number of vertices of the input graph). Using the parameterized complexity terminology, this implies that a huge number of graph problems are fixed-parameter tractable when parameterized by the branchwidth of their input graph. As the bounds for $f(\mathbf{bw})$ provided by Courcelle's theorem are huge, the design of tailor-made dynamic programming algorithms for specific problems, so that $f(\mathbf{bw})$ is a simple (preferably single exponential) function, became a natural (and unavoidable) ingredient for many papers on algorithms design (see [38, 57, 98, 188]).

Dynamic programming. Dynamic programming is applied in a bottom-up fashion on a rooted branch decomposition of the input graph, that roughly is a way to decompose the graph into a tree structure of edge bipartitions (the formal definition can be found in page 16). Each bipartition defines a separator of the graph called *middle set*, of cardinality bounded by the branchwidth of the input graph. The decomposition is routed in the sense that one of the parts of each bipartition is the "lower part of the middle set", i.e., the so-far processed one. For each graph problem, dynamic programming requires the suitable definition of tables encoding how potential (global) solutions of the problem are restricted in the middle set and the corresponding lower part. The size of these tables reflects the dependence of **bw** in the running time of the dynamic programming. Defining the tables is not always an easy task, as they depend on the particularities of each problem (some typical examples are shown in Section 8.3). In many cases, problems are grouped together according to the similarities in the way to treat them, and usually this leads to distinct upper bounds for the function $f(\mathbf{bw})$. We define the following categories of dynamic programming algorithms (below S denotes a the middle set of a branch decomposition):

(A) those where the tables encode a fixed number of *vertex subsets of* S;

(B) those where the tables encode a fixed number of *connected pairings of vertices of* S; and

(C) those where the tables encode a fixed number of *connected packings of* S *into sets*.

In Categories (B) and (C), by the term *connected* for the pairings (resp. packings) we mean that they correspond to a packing of paths (resp. trees) in the lower part of the middle set S. The above classification also induces a classification of graph problems depending on whether they can be solved by some algorithm in some of the above categories[2]. Notice that the problems in Category (A) belong also to Category (B), and problems in Category (B) are also problems in Category (C). Clearly, the size of the tables for problems in Category (A) is a single exponential function of the middle set size. Therefore, for such problems we have that $f(\mathbf{bw}) = 2^{\mathcal{O}(\mathbf{bw})}$. Such problems are, for instance, 3-Coloring, Vertex Cover, Dominating Set, or Independent Set (see Section I.2.4), whose common characteristic is that the certificate of the solution is a set (or a fixed number of

[1]The original statement of Courcelle's theorem used the parameter of treewidth instead of branchwidth. The two parameters are approximately equivalent, in the sense that the one is a constant factor approximation of the other (see page 16).

[2]To facilitate our presentation, we present in Section 8.3 the dynamic programming algorithms for a problem in Category (A) and a problem in Category (C).

sets) of vertices whose choice is not restricted by some global condition. Unfortunately, when connectivity conditions are applied, the tables of the dynamic programming are of size $2^{O(\mathbf{bw}\cdot\log(\mathbf{bw}))}$ or more. This happens because one needs to encode more information on the way a possible solution of the problem is situated in the middle set S, which usually classifies the problems in categories (B) or (C). Typical problems in Category (B) are Longest Path and Hamiltonian Cycle, where pairings correspond to the connected portions of a solution to the lower part of the middle set. Typical problems in Category (C) are Connected Vertex Cover and Maximum Induced Forest[3], where the connected portions of a solution may be identified by sets of arbitrary cardinality. For Category (B), the size of the tables is lower-bounded by the number of perfect matchings of a complete bipartite graph of k vertices, that is by $2^{\Theta(k\cdot\log k)}$. For Category (C), the size of the tables is lower bounded by the k-th Bell number, that is again lower-bounded by $2^{\Theta(k\cdot\log k)}$. In both cases, this implies algorithms where $f(\mathbf{bw}) = 2^{O(\mathbf{bw}\cdot\log \mathbf{bw})}$.

Single-exponentiality: results and techniques. The most desired characteristic of any dynamic programming algorithm is the single exponential dependence on the branch-width of the input graph (according to the results in [148], this dependence is optimal for many combinatorial problems). Exponential dependence is trivial for problems in Category (A), and may become possible for the other two categories when we take into account the *structural properties* of the input graph. For problems in Category (B), the first step in this direction was done in [100] for planar graphs (see Chapter 7 for extensions of this technique for problems in Category (C)). The idea in [100] is to use a special type of branch decomposition called *sphere cut* decomposition (introduced in [180], see page 168) that can guarantee that the pairings are non-crossing pairings (because of the connectivity demand) around a virtual edge-avoiding cycle (called *noose*) of the plane where G is embedded. This restricts the number of tables corresponding to a middle set of size k by the k-th Catalan number, which is *single-exponential* in k. That way, single-exponential (in **bw**) running times can be designed for problems in Category (B). The same approach was extended for graphs of Euler genus γ in [97]. The idea was to perform a *planarization* of the input graph by splitting the potential solution into at most γ pieces and then applying the sphere cut decomposition technique of [100] to a more general version of the problem where the number of pairings is still bounded by some Catalan number. This made it possible to avoid dealing with the combinatorial structures in surfaces, where the arrangement of the solutions are harder to handle. The same idea was applied in [99] for H-minor free graphs using much more involved Catalan structures, again for problems in Category (B).

Our results. In this chapter, we follow a different approach in order to design single exponential (in **bw**) algorithms for graphs embedded in surfaces. In particular, we deviate significantly from the planarization technique of [97]. Instead, we extend the concept of sphere cut decomposition from planar graphs to surfaces and we exploit directly the combinatorial structure of the potential solutions in the topological surface. Our approach

[3]Notice that the Maximum Induced Forest problem is equivalent to the Feedback Vertex Set problem. We choose this way to present it in order to make more visible its classification into Category (C).

permits us to provide combinatorial bounds for problems in Category (C). Apart from those mentioned above, examples of such problems are Maximum d-Degree-Bounded Connected Subgraph, Maximum d-Degree-Bounded Connected Induced Subgraph and all the variants studied in Chapter 7, Connected Dominating Set, Connected r-Domination, Connected FVS, Maximum Leaf Spanning Tree, Maximum Full-Degree Spanning Tree, Maximum Eulerian Subgraph, Steiner Tree, and Maximum Leaf Tree. As Category (C) includes the problems in Category (B), our results imply all the results in [97], and with running times whose genus dependence is better than the ones in [97], as discussed in Section 8.14.

Our techniques. Our analysis is based on a special type of branch decomposition of embedded graphs with nice topological properties, which we call *surface cut decomposition* (see Section 8.6). Roughly, the middle sets of such a decomposition are situated along a bounded (by the genus γ) set of nooses of the surface with few (again bounded by γ) common points. The construction of such a decomposition is based on the concept of *polyhedral decomposition* introduced in Section 8.4. In Section 8.5, we prove some basic properties of surface cut decompositions that make it possible to bound the sizes of the tables of the dynamic programming. They correspond to the number of non-crossing partitions of vertex sets laying in the boundary of a generic surface. To count these partitions, we use a powerful technique of analytic combinatorics: *singularity analysis* over expressions obtained by the *symbolic method* (for more on this technique, see the monograph of Flajolet and Sedgewick [115]). The symbolic method gives a precise asymptotic enumeration of the number of non-crossing partitions, that yields the single exponentiality of the table size (see Section 8.7). To solve a problem in Category (C), our approach resides on a common preprocessing step that is to construct the *surface cut decomposition* (Algorithm 2 in Section 8.6). Then, what remains is just to run the dynamic programming algorithm on such a surface cut decomposition. The exponential bound on the size of the tables of this dynamic programming algorithm is provided as a result of our analysis in Theorem 8.4 of Section 8.14.

Section 8.2 is devoted to provide all the preliminaries to read the sequel of the chapter.

8.2 Background and Notation

Sections 8.2.1, 8.2.2, 8.2.3, and 8.2.4 contain the basic background and the notation we will use concerning topological surfaces, graphs embedded in surfaces, carving decompositions and clique sums, and the symbolic method and analytic combinatorics, respectively.

8.2.1 Topological surfaces

In this chapter, we consider compact surfaces with boundary homeomorphic to a finite set (possibly empty) of disjoint circles. We denote the number of connected components of the boundary of a surface Σ by $\beta(\Sigma)$. The surface classification theorem asserts that a compact, connected and without boundary surface is determined, up to homeomorphism,

by its Euler characteristic $\chi(\Sigma)$ and by whether it is orientable or not. More precisely, orientable surfaces are obtained by adding $g \geq 0$ *handles* to the sphere $\mathbb{S}^2$, obtaining the g-torus $\mathbb{T}_g$ with Euler characteristic $\chi(\mathbb{T}_g) = 2-2g$, while non-orientable surfaces are obtained by adding $h > 0$ *cross-caps* to the sphere, hence obtaining a non-orientable surface $\mathbb{P}_h$ with Euler characteristic $\chi(\mathbb{P}_h) = 2 - h$. Writing $\overline{\Sigma}$ for the surface (without boundary) obtained from Σ by gluing a disk on each of the $\beta(\Sigma)$ components of the boundary, we obtain the equality $\chi(\overline{\Sigma}) = \beta(\Sigma) + \chi(\Sigma)$. A subset Π of a surface Σ is *surface-separating* if $\Sigma \setminus \Pi$ has at least 2 connected components.

As a conclusion, our surfaces are determined, up to homeomorphism, by their orientability, their Euler characteristic and the number of connected components of their boundary. For computational simplicity, it is convenient to work with the *Euler genus* $\gamma(\Sigma)$ of a surface Σ, which is defined as $\gamma(\Sigma) = 2 - \chi(\Sigma)$.

8.2.2 Graphs embedded in surfaces

Our main reference for graphs on surfaces is the monograph of Mohar and Thomassen [163]. For a graph G we use the notation (G, τ) to denote that τ is an embedding of G in Σ, whenever the surface Σ is clear from the context. An embedding has *vertices*, *edges*, and *faces*, which are 0, 1, and 2 dimensional open sets, and are denoted $V(G)$, $E(G)$, and $F(G)$, respectively. We use $e(G)$ to denote $|E(G)|$. In a *2-cell embedding*, also called *map*, each face is homeomorphic to a disk. The degree $\mathrm{d}(v)$ of a vertex v is the number of edges incident with v, counted with multiplicity (loops are counted twice). An edge of a map has two ends (also called *half-edges*), and either one or two sides, depending on the number of faces which is incident with. A map is *rooted* if an edge and one of its half-edges and sides are distinguished as the root-edge, root-end and root-side, respectively. Notice that the rooting of maps on orientable surfaces usually omits the choice of a root-side because the underlying surface is oriented and maps are considered up to orientation preserving homeomorphism. Our choice of a root-side is equivalent in the orientable case to the choice of an orientation of the surface. The root-end and -sides define the root-vertex and -face, respectively. The rooted maps are considered up to cell-preserving homeomorphisms preserving the root-edge, -end, and -side. In figures, the root-edge is indicated as an oriented edge pointing away from the root-end and crossed by an arrow pointing towards the root-side (this last, provides the orientation in the surface).

For a graph G, the *Euler genus* of G, denoted $\gamma(G)$, is the smallest Euler genus among all surfaces in which G can be embedded. Determining the Euler genus of a graph is an NP-hard problem [189], hence we assume throughout the paper that we are given an already embedded graph. An *O-arc* is a subset of Σ homeomorphic to $\mathbb{S}^1$. A subset of Σ meeting the drawing only at vertices of G is called *G-normal*. If an O-arc is G-normal, then we call it a *noose*. The *length* of a noose is the number of its vertices. Many results in topological graph theory rely on the concept of *representativity* [174, 180], also called *facewidth*, which is a parameter that quantifies local planarity and density of embeddings. The representativity $\mathbf{rep}(G, \tau)$ of a graph embedding (G, τ) is the smallest length of a non-contractible (i.e., non null-homotopic) noose in Σ. We call an embedding (G, τ) *polyhedral* [163] if G is 3-connected and $\mathbf{rep}(G, \tau) \geq 3$, or if G is a clique and $1 \leq |V(G)| \leq 3$. With abuse of notation, we also say in that case that the graph G itself is polyhedral.

For a given embedding (G,τ), we denote by (G^*,τ) its dual embedding. Thus G^* is the geometric dual of G. Each vertex v (resp. face r) in (G,τ) corresponds to some face v^* (resp. vertex r^*) in (G^*,τ). Also, given a set $S \subseteq E(G)$, we denote as S^* the set of the duals of the edges in S. Let (G,τ) be an embedding and let (G^*,τ) be its dual. We define the *radial graph embedding* (R_G,τ) of (G,τ) (also known as *vertex-face graph embedding*) as follows: R_G is an embedded bipartite graph with vertex set $V(R_G) = V(G) \cup V(G^*)$. For each pair $e = \{v,u\}$, $e^* = \{u^*,v^*\}$ of dual edges in G and G^*, R_G contains edges $\{v,v^*\}$, $\{v^*,u\}$, $\{u,u^*\}$, and $\{u^*,v\}$. Mohar and Thomassen [163] proved that, if $|V(G)| \geq 4$, the following conditions are equivalent: *(i)* (G,τ) is a polyhedral embedding; *(ii)* (G^*,τ) is a polyhedral embedding; and *(iii)* (R_G,τ) has no multiple edges and every 4-cycle of R_G is the border of some face. The *medial graph embedding* (M_G,τ) of (G,τ) is the dual embedding of the radial embedding (R_G,τ) of (G,τ). Note that (M_G,τ) is a Σ-embedded 4-regular graph.

8.2.3 Carving decompositions and clique sums

Recall the definition of branch decompositions and branchwidth from Section I.1.1 (page 16). A *carving decomposition* (T,μ) is similar to a branch decomposition, only with the difference that μ is a bijection between the leaves of the tree and the vertex set of the graph G. For an edge e of T, the counterpart of the middle set, called the *cut set* $\mathbf{cut}(e)$, contains the edges of G with endvertices in the leaves of both subtrees. The counterpart of branchwidth is *carvingwidth*, and is denoted by $\mathbf{cw}(G)$.

Let $G = (V,E)$ be a connected graph. For $S \subseteq V$, we denote by $\delta(S)$ the set of all edges with an end in S and an end in $V \setminus S$. Let $\{V_1,V_2\}$ be a partition of V. If $G[V \setminus V_1]$ and $G[V \setminus V_2]$ are both non-null and connected, we call $\delta(V_1)$ a *bond* of G [180]. In a *bond carving decomposition*, every cut set is a bond of the graph. That is, in a bond carving decomposition, every cut set separates the graph into two connected components.

We now recall the definition of clique sum (see page 154). Let G_1 and G_2 be graphs with disjoint vertex-sets and let $k \geq 0$ be an integer. For $i = 1,2$, let $W_i \subseteq V(G_i)$ form a clique of size k and let G_i' $(i = 1,2)$ be obtained from G_i by deleting some (possibly no) edges from $G_i[W_i]$ with both endpoints in W_i. Consider a bijection $h : W_1 \to W_2$. We define a *clique sum* G of G_1 and G_2, denoted by $G = G_1 \oplus_k G_2$, to be the graph obtained from the union of G_1' and G_2' by identifying w with $h(w)$ for all $w \in W_1$. The integer k is called the *size* of the clique sum.

8.2.4 The symbolic method and analytic combinatorics

The main reference in enumerative combinatorics is the monograph of Flajolet and Sedgewick [115]. The framework introduced in this book gives a language to translate combinatorial conditions between combinatorial classes into equations. This is what is called the *symbolic method* in combinatorics. Later, we can treat these equations as relations between analytic functions. This point of view gives us the possibility to use complex analysis techniques to obtain information about the combinatorial classes. This is the origin of the term *analytic combinatorics*.

For a set $\mathcal{A}$ of objects, let $|\cdot|$ be an application from $\mathcal{A}$ to $\mathbb{N}$, which is called the *size*. A pair $(\mathcal{A}, |\cdot|)$ is called a *combinatorial class*. Define the formal power series (called the *generating function* or *GF* associated to the class) $\mathbf{A}(z) = \sum_{a\in\mathcal{A}} z^{|a|} = \sum_{k=0}^{\infty} a_k z^k$. The constructions we use in this work and their translation into the language of GFs are shown in Table 8.1.

Construction		GF
Union	$\mathcal{A} \cup \mathcal{B}$	$\mathbf{A}(z) + \mathbf{B}(z)$
Product	$\mathcal{A} \times \mathcal{B}$	$\mathbf{A}(z)\mathbf{B}(z)$
Sequence	$\mathbf{Seq}(\mathcal{A})$	$\frac{1}{1-\mathbf{A}(z)}$
Pointing	$\mathcal{A}^\bullet$	$z\frac{\partial}{\partial z}\mathbf{A}(z)$

Table 8.1: Constructions and translations into GF.

The union $\mathcal{A} \cup \mathcal{B}$ of $\mathcal{A}$ and $\mathcal{B}$ refers to the disjoint union of the classes. The cartesian product $\mathrm{A} \times \mathcal{B}$ of $\mathcal{A}$ and $\mathcal{B}$ is the set $\{(a, b) : a \in \mathcal{A}, b \in \mathcal{B}\}$. The sequence $\mathbf{Seq}(\mathcal{A})$ of a set $\mathcal{A}$ corresponds to the set $\mathcal{E} \cup \mathcal{A} \cup \mathcal{A}\times\mathcal{A} \cup \mathcal{A}\times\mathcal{A}\times\mathcal{A} \cup \ldots$. At last, the pointing operator $\mathcal{A}^\bullet$ of a set $\mathcal{A}$ consists in pointing one of the atoms of each element $a \in \mathcal{A}$.

The study of the growth of the coefficients of GFs can be obtained by considering GFs as complex functions which are analytic around $z = 0$. This is the philosophy of analytic combinatorics. The growth behavior of the coefficients depends only on the smallest positive singularity of the GF. Its *location* provides the *exponential growth* of the coefficients, and its *behavior* gives the *subexponential growth* of the coefficients.

The basic results in this area are the so-called *Transfer Theorems for singularity analysis*. These results allows us to deduce asymptotic estimates of an analytic function using its asymptotic expansion near its dominant singularity. The precise statement is claimed in [115]. We use the following reduced version of the theorem (without taking care of technical conditions, which are satisfied in all cases): let $\mathbf{F}(z)$ be a GF with positive coefficients, such that ρ is its smallest real singularity. Suppose that $\mathbf{F}(z)$ admits a singular expansion around $z = \rho$ of the form $\mathbf{F}(z) =_{z\to\rho} C(1 - z/\rho)^{-\alpha} + O\left((1 - z/\rho)^{-\alpha+1/2}\right)$, where C is a constant. Then the Transfer Theorem for singularity analysis states that, for k big enough, the following estimate holds

$$[z^k]\mathbf{F}(z) =_{k\to\infty} C\frac{k^{\alpha-1}}{\Gamma(\alpha)}\rho^{-k}\left(1 + O\left(k^{-1/2}\right)\right). \tag{8.1}$$

8.3 Examples of Dynamic Programming Algorithms

In this section we present two examples of typical dynamic programming algorithms on graphs of bounded branchwidth. The first algorithm solves the Vertex Cover problem and belongs in Category (A) while he second solves the Connected Vertex Cover problem and belongs in Category (C) but not in (B) (nor in (A)).

The standard dynamic programming approach on branch decompositions requires the so-called *rooted* branch decomposition defined as a triple (T, μ, e_r) where (T, μ) is a branch-decomposition of G where T is a tree rooted on a leaf v_l incident to some edge e_r of T. We insist that no edge of G is assigned to v_l and thus $\mathbf{mid}(e_r) = \emptyset$ (for this, we take any edge of a branch decomposition, subdivide it and then connect by e_r the subdivision vertex with a new leaf t_l). The edges of T can be oriented towards the root e_r and for each edge $e \in E(T)$ we denote by E_e the edges of G that are mapped to leaves of T that are descendants of e. We also set $G_e = G[E_e]$ and we denote by $L(T)$ the edges of T that are incident to leaves

of T. Given an edge e heading at a non-leaf vertex v, we denote by $e_1, e_2 \in E(T)$ the two edges with tail v. As both examples below are variants of the vertex cover problem, we can also assume that $|E(T)| = O(k \cdot n)$ as, otherwise, the answer for each problem is trivially negative.

Dynamic programming for Vertex Cover**.** Let G be a graph and $X, X' \subseteq V(G)$ where $X \cap X' = \emptyset$. We say that $\mathbf{vc}(G, X, X') \leq k$ if G contains a vertex cover S where $|S| \leq k$ and $X \subseteq S \subseteq V(G) \setminus X'$. Let $\mathcal{R}_e = \{(X, k) \mid \mathbf{vc}(G_e, X, \mathbf{mid}(e) \setminus X) \leq k\}$ and observe that $\mathbf{vc}(G) \leq k$ iff $(\emptyset, k) \in \mathcal{R}_{e_r}$. For each $e \in E(T)$ we can compute $\mathcal{R}_e$ by using the following dynamic programming formula:

$$\mathcal{R}_e = \begin{cases} \{(X,k) \mid X \neq \emptyset \wedge k \geq |X|\} & \text{if } e \in L(T) \\ \{(X,k) \mid \exists (X_1,k_1) \in \mathcal{R}_{e_1}, \exists (X_2,k_2) \in \mathcal{R}_{e_2} : \\ (X_1 \cup X_2) \cap \mathbf{mid}(e) = X \wedge k_1 + k_2 - |X_1 \cap X_2| \leq k\} & \text{if } e \notin L(T) \end{cases}$$

Notice that for each $e \in E(T)$, $|\mathcal{R}_e| \leq 2^{|\mathbf{mid}(e)|} \cdot k$. Therefore, the above algorithm can check whether $\mathbf{vc}(G) \leq k$ in $O(2^{\mathbf{bw}(G)} \cdot k \cdot |V(T)|)$ steps. Clearly, this simple algorithm is single exponential on $\mathbf{bw}(G)$. Moreover the above dynamic programming machinery can be adapted to many other combinatorial problems where the certificate of the solution is a (non-restricted) subset of vertices (e.g. Dominating Set, 3-Coloring, Independent Set, among others).

Dynamic programming for Connected Vertex Cover**.** Suppose now that we are looking for a *connected* vertex cover of size $\leq k$. Clearly, the above dynamic programming formula does not work for this variant as we should book-keep more information on X towards encoding the connectivity demand.

Let G be a graph, $X \subseteq V(G)$ and $\mathcal{H}$ be a (possibly empty) hypergraph whose vertex set is a subset of X, whose hyperedges are non-empty, pairwise non-intersecting, and such that each vertex of $\mathcal{H}$ belongs to some of its hyperedges (we call such a hypergraph *partial packing* of X). Suppose that $\mathcal{H}$ is a partial packing on $\mathbf{mid}(e)$. We say that $\mathbf{cvc}(G, \mathcal{H}) \leq k$ if G contains a vertex cover S where $|S| \leq k$ and such that if $\mathcal{C}$ is the collection of the connected components of $G_e[S]$, then either $|E(\mathcal{H})| = |\mathcal{C}|$ and $(X, \{X \cap V(C) \mid C \in \mathcal{C}\}) = \mathcal{H}$ or $E(\mathcal{H}) = \emptyset$ and $|\mathcal{C}| = 1$.

As before, let $\mathcal{Q}_e = \{(\mathcal{H}, k) \mid \mathbf{cvc}(G, \mathcal{H}) \leq k\}$ and observe that $\mathbf{cvc}(G) \leq k$ iff $((\emptyset, \emptyset), k) \in \mathcal{Q}_{e_r}$. The dynamic programming formula for computing $\mathcal{Q}_e$ for each $e \in E(T)$ is the following.

$$\mathcal{Q}_e = \begin{cases} \{(\mathcal{H},k) \mid \min\{k, |E(\mathcal{H})|+1\} \geq |V(\mathcal{H})| \geq 1 & \text{if } e \in L(T) \\ \{(\mathcal{H},k) \mid \exists (\mathcal{H}_1,k_1) \in \mathcal{Q}_{e_1}, \exists (\mathcal{H}_2,k_2) \in \mathcal{Q}_{e_2} : \\ V(\mathcal{H}_1) \cap (\mathbf{mid}(e_1) \cap \mathbf{mid}(e_2)) = V(\mathcal{H}_2) \cap (\mathbf{mid}(e_1) \cap \mathbf{mid}(e_2)), \\ (\mathcal{H}_1 \oplus \mathcal{H}_2)[\mathbf{mid}(e)] = \mathcal{H}, k_1 + k_2 - |V(\mathcal{H}_1) \cap V(\mathcal{H}_2)| \leq k\}, \\ \text{if } E(\mathcal{H}) = \emptyset \text{ then } |E(\mathcal{H}_1 \oplus \mathcal{H}_2)| = 1, \text{and} \\ \text{if } E(\mathcal{H}) \neq \emptyset \text{ then } |E(H_1 \oplus H_2)| = |E(\mathcal{H})| & \text{if } e \notin L(T). \end{cases}$$

In the above formula, $\mathcal{H}_1 \oplus \mathcal{H}_2$ is the hypergraph with vertex set $V(\mathcal{H}_1) \cup V(\mathcal{H}_2)$ where each of its hyperedges contains the vertices of each of the connected components of $\mathcal{H}_1 \cup \mathcal{H}_2$.

Clearly each $\mathcal{H}$ corresponds to a collection of subsets of X and the number of such collections for a given set $\mathbf{mid}(e)$ of r elements is given by the r-th Bell number of r, denoted by B_r. By taking the straightforward upper bound $|B_r| = 2^{O(r \log r)}$, we have that one can check whether an input graph G has a connected vertex cover of size at most k in $O(2^{\mathbf{bw}(G)\cdot\log(\mathbf{bw}(G))} \cdot k \cdot |V(T)|)$ steps.

As the growth of B_r is not single exponential, we cannot hope for a single exponential (in $\mathbf{bw}(G)$) running time for the above dynamic programming procedure and no algorithm is known for this problem that runs in time that is single exponential in $\mathbf{bw}(G)$. The same problem appears for numerous other problems where further restrictions apply to their solution certificates. Such problems can be connected variants of problems in Category (A) and others such as Maximum Induced Forest, Maximum d-Degree-Bounded Connected Subgraph, Maximum d-Degree-Bounded Connected Induced Subgraph and all the variants studied in Chapter 7, Connected Dominating Set, Connected r-Domination, Connected FVS, Maximum Leaf Spanning Tree, Maximum Full-Degree Spanning Tree, Maximum Eulerian Subgraph, Steiner Tree, or Maximum Leaf Tree.

8.4 Polyhedral Decompositions

We introduce in this section *polyhedral decompositions* of graphs embedded in surfaces. Let G be an embedded graph, and let N be a noose in the surface. Similarly to [62], we use the notation $G \rtimes N$ for the graph obtained by cutting G along the noose N and gluing a disk on the obtained boundaries.

Definition 8.1 *Given a graph $G = (V, E)$ embedded in a surface of Euler genus γ, a* polyhedral decomposition *of G is a set of graphs $\mathcal{G} = \{H_1, \ldots, H_\ell\}$ together with a set of vertices $A \subseteq V$ such that*

- $|A| = O(\gamma)$;
- *H_i is a minor of $G[V \setminus A]$, for $i = 1, \ldots, \ell$;*
- *H_i has a polyhedral embedding in a surface of Euler genus at most γ, for $i = 1, \ldots, \ell$; and*
- *$G[V \setminus A]$ can be constructed by joining the graphs of $\mathcal{G}$ applying clique sums of size $0, 1,$ or 2.*

Remark 8.1 *Note that an embedded graph H is not polyhedral if and only if there exists a noose N of length at most 2 in the surface in which H is embedded, such that either N is non-contractible or $V(H) \cap N$ separates H. Indeed, if H has representativity at most 2, then there exists a non-contractible noose N of length at most 2. Otherwise, since H is not polyhedral, H has a minimal separator S of size at most 2. It is then easy to see that there exists a noose containing only vertices of S.*

Algorithm 1 Construction of a polyhedral decomposition of an embedded graph G

Input: A graph G embedded in a surface of Euler genus γ.
Output: A polyhedral decomposition of G.

$A = \emptyset$, $\mathcal{G} = \{G\}$ (the elements in $\mathcal{G}$, which are embedded graphs, are called *components*).
while $\mathcal{G}$ contains a non-polyhedral component H **do**
  Let N be a noose as described in Observation 8.1 in the surface in which H is embedded, and let $S = V(H) \cap N$.
  if N is non-surface-separating **then**
    Add S to A, and replace in $\mathcal{G}$ component H with $H[V(H) \setminus S] \rtimes N$.
  end if
  if N is surface-separating **then**
    Let H_1, H_2 be the subgraphs of $H \rtimes N$ corresponding to the two surfaces occurring after splitting H
    if $S = \{u\} \cup \{v\}$ and $\{u, v\} \notin E(H)$ **then**
      Add the edge $\{u, v\}$ to H_i, $i = 1, 2$.
    end if
    Replace in $\mathcal{G}$ component H with the components of $H \rtimes N$ containing at least one edge of H.
  end if
end while
return $\{\mathcal{G}, A\}$.

Algorithm 1 provides an efficient way to construct a polyhedral decomposition, as it is stated in Proposition 8.1.

In the above algorithm, the addition of an edge $\{u, v\}$ represents the existence of a path in G between u and v that is not contained in the current component.

Proposition 8.1 *Given a graph G on n vertices embedded in a surface, Algorithm 1 constructs a polyhedral decomposition of G in $O(n^3)$ steps.*

Proof: We first prove that the the output $\{\mathcal{G}, A\}$ of Algorithm 1 is indeed a polyhedral decomposition of G, and then we analyze the running time.

Let us see that each component of $\mathcal{G}$ is a minor of $G[V \setminus A]$. Indeed, the only edges added to G by Algorithm 1 are those between two non-adjacent vertices u, v that separate a component H into several components $H_1, \ldots, H_\ell$. For each component H_i, $i = 1, \ldots, \ell$, there exists a path between u and v in $H \setminus H_i$ (provided that the separators of size 1 have been already removed, which can we assumed w.l.o.g.), and therefore the graph obtained from H_i by adding the edge $\{u, v\}$ is a minor of H, which is inductively a minor of $G[V \setminus A]$. Also, each component of $\mathcal{G}$ is polyhedral by definition of the algorithm.

As a non-separating noose is necessarily non-contractible, each time some vertices are moved to A the Euler genus of the surfaces strictly decreases [163, Lemma 4.2.4]. Therefore, $|A| = O(\gamma)$.

By the construction of the algorithm, it is also clear that each component of $\mathcal{G}$ has a polyhedral embedding in a surface of Euler genus at most γ. Finally, $G[V \setminus A]$ can be constructed by joining the graphs of $\mathcal{G}$ applying clique sums of size $0, 1$, or 2.

Thus, $\{\mathcal{G}, A\}$ is a polyhedral decomposition of G by Definition 8.1.

We now analyze the running time of the algorithm. Separators of size at most 2 can be found in $O(n^2)$ steps [142]. A noose with respect to a graph H corresponds to a cycle in the radial graph of H, hence can also be found[4] in $O(n^2)$ (using that the number of edges of a bounded-genus graph is linearly bounded by its number of vertices). Since each time that we find a *small* separator we decrease the size of the components, the running time of the algorithm is $O(n^3)$. □

8.5 Some Topological Lemmata

In this section we state some topological lemmata that are used in Sections 8.6 and 8.7. In particular, Lemmata 8.1 and 8.2 are used in the proof of Theorem 8.1 (page 189) while Lemma 8.3 is used in the proof of Theorem 8.2 (page 196).

Given a graph G embedded in a surface of Euler genus γ, its dual G^* and a spanning tree C^* of G^*, we call $C = \{e \in E(G) \mid e^* \in E(C^*)\}$ a *spanning cotree* of G. We define a *tree-cotree partition* (cf. [107]) of an embedded graph G to be a triple (T, C, X) where T is a spanning tree of G, C is a spanning cotree of G, $X \subseteq E(G)$, and the three sets $E(T)$, C, and X form a partition of $E(G)$. Eppstein proved [107, Lemma 3.1] that if T and C^* are forests such that $E(T)$ and C are disjoint, we can make T become part of a spanning tree T' and C become part of a spanning cotree disjoint from T', extending T and C to a tree-cotree decomposition. We can now announce the following lemma from [107, Lemma 3.2].

Lemma 8.1 (Eppstein [107]) *If (T, C, X) is a tree-cotree decomposition of a graph G embedded in a surface of Euler genus γ, then $|X| = O(\gamma)$.*

Lemma 8.2 *Let Σ be a surface without boundary. Let S be a set of (not necessary disjoint) $O(\gamma(\Sigma))$ cycles such that $\Sigma \backslash S$ has 2 connected components. Let H be the graph corresponding to the union of the cycles in S. Then $\sum_{v \in V(H)} (\mathrm{d}(v) - 2) = O(\gamma(\Sigma))$.*

Proof: Let κ be the number of cycles, so by assumption $\kappa = O(\gamma(\sigma))$. The first step consists in simplifying the problem. Let H' be the graph obtained from H by dissolving vertices of degree 2 (except from the case of isolated cycles, where we obtain a single vertex of degree 2 and a loop). Each dissolved vertex had degree 2, so the sum $\sum_{v \in V(H')} (\mathrm{d}(v) - 2)$ coincides with $\sum_{v \in V(H)} (\mathrm{d}(v) - 2)$. Additionally, by assumption H' separates Σ into 2 connected components Σ' and Σ''. Let $H'_1, H'_2, \ldots, H'_r$ be the maximal connected subgraphs of H'. In particular, $r \leq \kappa$.

Some of these connected subgraphs may be incident with Σ' but not with Σ'', or conversely. Additionally, there is at least one connected subgraph H'_i incident with both connected components. W.l.o.g. we assume that the subgraphs $H'_1, H'_2, \ldots, H'_p$ are incident only with Σ', $H'_{p+1}, \ldots, H'_q$ are incident with both components, and $H'_{q+1}, \ldots, H'_r$ are incident only with Σ''. It is obvious that there exists a path joining a vertex of H'_i with a vertex of H'_{i+1} if $1 \leq i \leq q-1$ or $p+1 \leq i \leq r-1$.

[4] A shortest non-contractible cycle can be found in $O(2^{O(\gamma \log \gamma)} n^{4/3})$ steps [62]. This time improves on $O(n^3)$ for a big range of values of γ.

From graphs $H'_1, H'_2, \ldots, H'_p, \ldots, H'_q$ (the ones which are incident with Σ') we construct a new graph G_1 in the following inductive way: we start taking H'_q and H'_{q-1}, and a path joining a vertex of H'_q to a vertex of H'_{q-1}. This path exists because H'_q and H'_{q-1} are incident with Σ'. Consider the graph obtained from H'_q and H'_{q-1} by adding an edge that joins this pair of vertices. Then, delete H'_q and H'_{q-1} from the initial list and add this new connected graph. This procedure is done $q-1$ times. At the end, we obtain a connected graph G' incident with both Σ' and Σ'' where each vertex has degree at least 3. Finally, we apply the same procedure with $G', H'_{q+1}, \ldots, H'_r$, obtaining a connected graph G. Observe also that

$$\sum_{v \in V(H)} (\mathrm{d}(v) - 2) \leq \sum_{v \in V(G)} (\mathrm{d}(v) - 2) < \sum_{v \in V(G)} \mathrm{d}(v) = 2|E(G)|.$$

In what follows, we obtain upper bounds for $2|E(G)|$. Observe that H' defines a pair of faces over Σ, not necessarily disks. In the previous construction of G, every time we add an edge either we subdivide a face into two parts or not. Consequently, the number of faces that G defines over Σ is at most $2+\kappa$. The next step consists in reducing the surface in the following way: let f be a face determined by G over Σ. If f is contractible, we do nothing. If not, there is a non-contractible cycle $\mathbb{S}^1$ contained on f. Consider the connected component of $\Sigma \rightarrowtail \mathbb{S}^1$ which contains G (call it Σ_1). It is obvious that G defines a decomposition of Σ_1, $\gamma(\Sigma_1) \leq \gamma(\Sigma)$, and the number of faces has been increased by at most one. Observe that for each operation $\rightarrowtail$ we reduce the Euler genus and we create at most one face. As the Euler genus is finite, the number of $\rightarrowtail$ operations is also finite. This gives rise to a surface Σ_s, where $s \leq \gamma(\Sigma)$, such that all faces determined by G are contractible. Additionally, the number of faces that G determines over Σ_s is smaller than $2 + \kappa + \gamma(\Sigma)$.

G defines a map on Σ_s (i.e., all faces are contractible), and consequently we can apply Euler's formula. Then $|F(G)| + |V(G)| = |E(G)| + 2 - \gamma(\Sigma_s)$. Then, $|F(G)| \leq 2 + \kappa + \gamma(\Sigma)$, so $|E(G)| + 2 - \gamma(\Sigma_s) = |V(G)| + |F(G)| \leq |V(G)| + 2 + \kappa + \gamma(\Sigma)$. The degree of each vertex is at least 3, thus $3|V(G)| \leq 2|E(G)|$. Substituting this condition in the previous equation, we obtain

$$|E(G)| + 2 - \gamma(\Sigma_s) \leq |V(G)| + 2 + \kappa + \gamma(\Sigma) \leq \frac{2}{3}|E(G)| + 2 + \kappa + \gamma(\Sigma).$$

Isolating $|E(G)|$, we get that $2|E(G)| \leq 6\kappa + 6\gamma(\Sigma_s) + 6\gamma(\Sigma) \leq 6\kappa + 12\gamma(\Sigma)$. This bound immediately translates into the statement of Lemma 8.2. □

We need another topological result, which deals with cycles that separate a given surface, and whose proof is an immediate consequence of [163, Proposition 4.2.1].

Lemma 8.3 *Let Σ be a surface with boundary and let $\mathbb{S}^1$ be a separating cycle. Let V_1 and V_2 be the connected components of $\Sigma \rightarrowtail \mathbb{S}^1$. Then $\gamma(\Sigma) = \gamma(V_1) + \gamma(V_2)$.*

8.6 Surface Cut Decompositions

Sphere cut decompositions [180] (see page 168) have proved to be very useful to analyze the running time of algorithms based on dynamic programming over branch decompositions on planar graphs [C17, 98–100]. In this section we generalize sphere cut decompositions to

graphs on surfaces; we call them *surface cut decompositions*. First we need a topological definition. A subset Π of a surface Σ is *fat-connected* if for every two points $p, q \in \Pi$, there exists a path $P \subseteq \Pi$ such that for every $x \in P$, $x \neq p, q$, there exists a subset D homeomorphic to an open disk such that $x \in D \subseteq \Pi$. We can now define the notion of surface cut decomposition.

Definition 8.2 *Given a graph G embedded in a surface Σ, a* surface cut decomposition *of G is a branch decomposition (T, μ) of G such that, for each edge $e \in E(T)$, there is a subset of vertices $A_e \subseteq V(G)$ with $|A_e| = O(\gamma(\Sigma))$ and either*

- $|\mathbf{mid}(e) \setminus A_e| \leq 2$, *or*
- *there exists a polyhedral decomposition $\{\mathcal{G}, A\}$ of G and a graph $H \in \mathcal{G}$ such that*
  - $A_e \subseteq A$;
  - $\mathbf{mid}(e) \setminus A_e \subseteq V(H)$;
  - *the vertices in $\mathbf{mid}(e) \setminus A_e$ are contained in a set $\mathcal{N}$ of $O(\gamma(\Sigma))$ nooses, such that the total number of occurrences in $\mathcal{N}$ of the vertices in $\mathbf{mid}(e) \setminus A_e$ is $|\mathbf{mid}(e) \setminus A_e| + O(\gamma(\Sigma))$; and*
  - $\Sigma \setminus \bigcup_{N \in \mathcal{N}} N$ *contains exactly two connected components, which are both fat-connected.*

Note that a sphere cut decomposition is a particular case of a surface cut decomposition when $\gamma = 0$, by taking $A_e = \emptyset$, $\mathcal{G}$ containing only the graph itself, and all the vertices of each middle set contained in a single noose.

We need some definitions and auxiliary results to be applied for building surface cut decompositions. In the same spirit of [125, Theorem 1] we can prove the following lemma. We omit the proof here since the details are very similar[5] to the proof in [125].

Lemma 8.4 *Let (G, τ) and (G^*, τ) be dual polyhedral embeddings in a surface of Euler genus γ and let (M_G, τ) be the medial graph embedding. Then* $\max\{\mathbf{bw}(G), \mathbf{bw}(G^*)\} \leq \mathbf{cw}(M_G)/2 \leq 6 \cdot \mathbf{bw}(G) + 2\gamma + O(1)$. *In addition, given a branch decomposition of G of width at most k, a carving decomposition of M_G of width at most $12k$ can be found in linear time.*

Lemma 8.5 (folklore) *The removal of a vertex from a non-acyclic graph decreases its branchwidth by at most* 1.

Lemma 8.6 *Let G be a graph and let $\mathcal{G}$ be a collection of graphs such that G can be constructed by joining graphs in $\mathcal{G}$ applying clique sums of size 0, 1, or 2. Given branch decompositions $\{(T_H, \mu_H) \mid H \in \mathcal{G})\}$, we can compute in linear time a branch decomposition (T, μ) of G such that $\mathbf{w}(T, \mu) \leq \max\{2, \{\mathbf{w}(T_H, \mu_H) \mid H \in \mathcal{G}\}\}$. In particular,* $\mathbf{bw}(G) \leq \max\{2, \{\mathbf{bw}(H) \mid H \in \mathcal{G}\}\}$.

Proof: Note that if G_1 and G_2 are graphs with no vertex (resp. a vertex, an edge) in common, then $G_1 \cup G_2 = G_1 \oplus_0 G_2$ (resp. $G_1 \oplus_1 G_2$, $G_1 \oplus_2 G_2$). To prove Lemma 8.6, we need the following two lemmata.

[5]The improvement in the multiplicative factor of the Euler genus is obtained by applying more carefully Euler's formula in the proof analogous to that of [125, Lemma 2].

Lemma 8.7 *Let G_1 and G_2 be graphs with at most one vertex in common. Then* $\mathbf{bw}(G_1 \cup G_2) = \max\{\mathbf{bw}(G_1), \mathbf{bw}(G_2)\}$.

Proof: Assume first that G_1 and G_2 share one vertex v. Clearly $\mathbf{bw}(G_1 \cup G_2) \geq \max\{\mathbf{bw}(G_1), \mathbf{bw}(G_2)\}$. Conversely, for $i = 1, 2$, let (T_i, μ_i) be a branch decomposition of G_i such that $\mathbf{w}(T_i, \mu_i) \leq k$. For $i = 1, 2$, let T_i^v be the minimal subtree of T_i containing all the leaves u_i of T_i such that v is an endpoint of $\mu_i(u_i)$. For $i = 1, 2$, we take an arbitrary edge $\{a_i, b_i\}$ of T_i^v, we subdivide it by adding a new vertex w_i, and then we build a tree T from T_1 and T_2 by adding the edge $\{w_1, w_2\}$. We claim that $(T, \mu_1 \cup \mu_2)$ is a branch decomposition of $G_1 \cup G_2$ of width at most k. Indeed, let us compare the middle sets of $(T, \mu_1 \cup \mu_2)$ to those of (T_1, μ_1) and (T_2, μ_2). First, it is clear that the vertices of $V(G_1) \cup V(G_2) - \{v\}$ appear in $(T, \mu_1 \cup \mu_2)$ in the same middle sets as in (T_1, μ_1) and (T_2, μ_2). Secondly, $\mathbf{mid}(\{w_1, w_2\}) = \{v\}$, since v is a cut-vertex of $G_1 \cup G_2$. Also, for $i = 1, 2$, $\mathbf{mid}(\{a_i, w_i\}) = \mathbf{mid}(\{w_i, b_i\}) = \mathbf{mid}(\{a_i, b_i\})$, and the latter has size at most k as $\mathbf{w}(T_i, \mu_i) \leq k$. For all other edges e of T_i, $i = 1, 2$, $\mathbf{mid}(e)$ is exactly the same in T and in T_i, since if $e \in E(T_i^v)$ then $v \in \mathbf{mid}(e)$ in both T and T_i, and if $e \in E(T_i \setminus T_i^v)$ then $v \notin \mathbf{mid}(e)$ in both T and T_i.

If G_1 and G_2 share no vertices, we can merge two branch decompositions (T_1, μ_1) and (T_2, μ_2) by subdividing a pair of arbitrary edges, without increasing the width. □

Lemma 8.8 ([124]) *Let G_1 and G_2 be graphs with one edge f in common. Then* $\mathbf{bw}(G_1 \cup G_2) \leq \max\{\mathbf{bw}(G_1), \mathbf{bw}(G_2), 2\}$. *Moreover, if both endpoints of f have degree at least 2 in at least one of the graphs, then* $\mathbf{bw}(G_1 \cup G_2) = \max\{\mathbf{bw}(G_1), \mathbf{bw}(G_2)\}$.

It remains only to show how to merge the branch decompositions (T_1, μ_1), (T_2, μ_2) of two graphs H_1, H_2 in $\mathcal{G}$. We distinguish four cases:

(a) H_1 and H_2 share two vertices v_1, v_2, and the edge $e = \{v_1, v_2\} \in E(G)$. We take the leaves in T_1 and T_2 corresponding to e, we identify them, and we add a new edge whose leave corresponds to e (see Figure 8.1(a)).

(b) H_1 and H_2 share two vertices v_1, v_2, and the edge $e = \{v_1, v_2\} \notin E(G)$. We take the leaves in T_1 and T_2 corresponding to e, we identify them, and we dissolve the common vertex (see Figure 8.1(b)).

(c) H_1 and H_2 share one vertex v. We take two edges b, c in T_1, T_2 whose leaves correspond to edges containing v, we subdivide them and add a new edge between the newly created vertices (see Figure 8.1(c)).

(d) H_1 and H_2 share no vertices. We do the construction of case (c) for any two edges of the two branch decompositions.

The above construction does not increase the branchwidth by Lemmata 8.7 and 8.8. □

Given an embedded graph G and a carving decomposition (T, μ) of its medial graph M_G, we define a *radial decomposition* (T^*, μ^*) of the dual graph R_G, where $T^* = T$ and μ^* is a bijection from the leaves of T to the set of faces of R_G defined as follows: for each edge

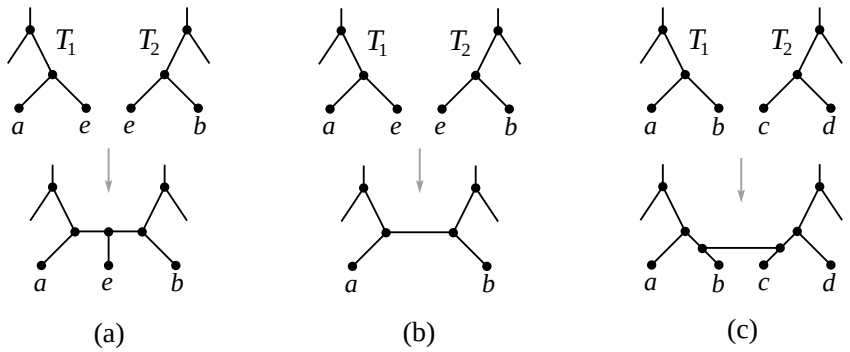

Figure 8.1: Merging branch decompositions (T_1, μ_1) and (T_2, μ_2) of two components H_1 and H_2 in a polyhedral decomposition $\{\mathcal{G}, A\}$ of $G = (V, E)$. There are three cases: (a) H_1 and H_2 share two vertices v_1, v_2 and the edge $e = \{v_1, v_2\}$ is in E; (b) H_1 and H_2 share two vertices v_1, v_2 and $e = \{v_1, v_2\}$ is *not* in E; (c) H_1 and H_2 share one vertex v.

$e \in E(T)$, $\mu^*(e) = f$, where f is the face in R_G corresponding to the vertex $u_f \in V(M_G)$ such that $\mu(e) = u_f$. Each edge $e \in E(T^*)$ partitions the faces of R_G into two sets F_1 and F_2. We define the *border set* of e, denoted $\mathbf{bor}(e)$, as the set of edges of R_G that belong to both F_1 and F_2. Note that F_1 and F_2 may intersect also in vertices, not only in edges. If (T, μ) is a bond carving decomposition of M_G, then the associated radial decomposition (also called *bond*) has nice connectivity properties. Indeed, in a bond carving decomposition, every cut set partitions the vertices of M_G into two subsets V_1, V_2 such that both $M_G[V_1]$ and $M_G[V_2]$ are non-null and connected. This property, seen in the radial decomposition of R_G, implies that each edge $e \in E(T^*)$ corresponds to a partition of the faces of R_G into two sets F_1 and F_2, namely *black* and *white* faces (naturally partitioning the edges into *black*, *white*, and *grey*), such that it is possible to reach any black (resp. white) face from any black (resp. white) face only crossing black (resp. white) edges. In other words, each set of monochromatic faces is fat-connected.

Remark 8.2 *Recall that all the faces of a radial graph R_G are* tiles*, that is, each face has exactly 4 edges. Also, each one of those tiles corresponds to a pair of dual edges e and e^* of G and G^*, respectively. Given a carving decomposition (T, μ) of M_G (or equivalently, a radial decomposition (T^*, μ^*) of R_G), one can obtain in a natural way branch decompositions of G and G^* by redefining the bijection μ from the leaves of T to the edges of G (or G^*) that correspond to the faces of R_G.*

We provide now an algorithm to construct a surface graph decomposition of an embedded graph. The proof of Theorem 8.1 uses Proposition 8.1, topological Lemmata 8.1 and 8.2, and the results of the current section.

Theorem 8.1 *Given a G on n vertices embedded in a surface of Euler genus γ, with $\mathbf{bw}(G) \leq k$, Algorithm 2 constructs, in $2^{3k+O(\log k)}n^3$ steps, a surface cut decomposition (T, μ) of G of width at most $27k + O(\gamma)$.*

Proof: We prove, in this order, that (1) the output (T, μ) of Algorithm 2 is indeed a surface cut decomposition of G; (2) the width of (T, μ) is at most $27\mathbf{bw}(G) + O(\gamma)$; and (3) the claimed running time.

Algorithm 2 Construction of a surface cut decomposition of an embedded graph G

Input: An embedded graph G.
Output: A surface cut decomposition of G.

Compute a polyhedral decomposition $\{\mathcal{G}, A\}$ of G, using Algorithm 1.
for each component H of $\mathcal{G}$ **do**
1. Compute a branch decomposition (T'_H, μ'_H) of H, using [35, Theorem 3.8].
2. Transform (T'_H, μ'_H) to a carving decomposition (T^c_H, μ^c_H) of the medial graph M_H, using Lemma 8.4.
3. Transform (T^c_H, μ^c_H) to a *bond* carving decomposition (T^b_H, μ^b_H) of M_H, using [180].
4. Transform (T^b_H, μ^b_H) to a branch decomposition (T_H, μ_H) of H, using Observation 8.2.

end for
Construct a branch decomposition (T, μ) of G by merging, using Lemma 8.6, the branch decompositions $\{(T_H, \mu_H) \mid H \in \mathcal{G}\}$, and by adding the set of vertices A to all the middle sets.
return (T, μ).

(1) (T, μ) is a surface cut decomposition of G.

We shall prove that all the properties of Definition 8.2 are fulfilled. For each $e \in E(T)$ we set $A_e = A \cap \mathbf{mid}(e)$, where A is the set of vertices output by Algorithm 1. Hence, by Proposition 8.1, $|A| = O(\gamma)$.

By construction, it is clear that (T, μ) is a branch decomposition of G. In (T, μ), there are some edges that have been added in the last step of Algorithm 2, in order to merge branch decompositions of the graphs in $\mathcal{G}$, with the help of Lemma 8.6. Let e be such an edge. Since $\{\mathcal{G}, A\}$ is a polyhedral decomposition of G, any pair of graphs in $\mathcal{G}$ share at most 2 vertices, hence $|\mathbf{mid}(e) \setminus A_e| \leq 2$.

All other edges of (T, μ) correspond to an edge of a branch decomposition of some polyhedral component $H \in \mathcal{G}$. Let henceforth e be such an edge. Therefore, $\mathbf{mid}(e) \setminus A_e \subseteq V(H)$. To complete this part of the proof, we prove in a sequence of three claims that the remaining conditions of Definition 8.2 hold.

Claim 8.1 *The vertices in* $\mathbf{mid}(e) \setminus A_e$ *are contained in a set* $\mathcal{N}$ *of* $O(\gamma)$ *nooses.*

Proof: The proof uses the tree-cotree partition defined in Section 8.5.

Recall that e is an edge that corresponds to a branch decomposition (T_H, μ_H) of a polyhedral component H of $\mathcal{G}$. The branch decomposition (T_H, μ_H) of H has been built by Algorithm 2 from a bond carving decomposition of its medial graph M_H, or equivalently from a bond radial decomposition of its radial graph R_H. Due to the fact that the carving decomposition of M_H is bond, edge e partitions the vertices of M_H into two sets – namely, *black* and *white* vertices – each one inducing a connected subgraph of M_H. There are three types of edges: *black*, *white*, and *grey*, according to whether they belong to faces of the same color (black or white) or not. Therefore, the corresponding black and white faces also induce connected subgraphs of R_H, in the sense that it is always possible to reach any black (resp. white) face from any black (resp. white) face only crossing black (resp. white) edges.

Let us now see which is the structure of the subgraph of R_H induced by the edges F belonging to both black and white faces look like. Since each edge of R_H contains a

vertex of H and another from H^*, the vertices in $\mathbf{mid}(e)$ are contained in $R_H[F]$, so it suffices to prove that $R_H[F]$ can be partitioned into a set of $O(\gamma)$ cycles (possibly sharing some vertices).

To this end, first note that in $R_H[F]$ all vertices have even degree. Indeed, let $v \in V(R_H[F])$, and consider a clockwise orientation of the edges incident to v in $R_H[F]$. Each of such edges alternates from a black to a white face, or viceversa, so beginning from an arbitrary edge and visiting all others edges in the clockwise order, we deduce that the number of edges incident to v is necessarily even.

Therefore, $R_H[F]$ can be partitioned into a set of cycles. Let us now bound the number of such cycles. For simplicity, let us identify a pair of dual edges e and e^* as the same edge. Since the subgraph induced by the black (resp. white) faces is connected, we can consider a spanning tree T_B^* (resp. T_W^*) of the black (resp. white) faces. Merge both trees by adding an edge e_0^*, and let T^* be the resulting tree. Let T be a spanning tree of the dual graph disjoint from T^* (such a spanning tree exists by [107, Lemma 3.1]). Now consider the tree-cotree partition (T, T^*, X), where X is the set of edges of R_H that are neither in T nor in T^*.

The edges of T^*, except e_0^*, separate faces of the same color. Therefore, the set $F \in E(R_H)$ of edges separating faces of different color is contained in $T \cup \{e_0\} \cup X$. Since T is a tree, each cycle of $R_H[F]$ uses at least one edge in $\{e_0\} \cup X$. Therefore, $R_H[F]$ can be partitioned into at most $1 + |X|$ cycles. The result follows from the fact that (T, T^*, X) is a tree-cotree partition and therefore $|X| = O(\gamma)$ by Lemma 8.1. □

Claim 8.2 *$\bigcup_{N \in \mathcal{N}} N$ separates Σ into 2 fat-connected components.*

Proof: By Claim 8.1, the vertices in $\mathbf{mid}(e) \setminus A_e$ are contained in $\bigcup_{N \in \mathcal{N}} N$. The claim holds from the fact that for each component H of $\mathcal{G}$, (T_H^b, μ_H^b) is a bond carving decomposition of M_H, and by taking into account the discussion before Observation 8.2 in Section 8.6. □

Claim 8.3 *The total number of occurrences in $\mathcal{N}$ of the vertices in $\mathbf{mid}(e) \setminus A_e$ is $|\mathbf{mid}(e) \setminus A_e| + O(\gamma)$.*

Proof: By Claim 8.2, $\bigcup_{N \in \mathcal{N}} N$ separates Σ into 2 fat-connected components. Let H be the graph induced in Σ by the nooses in $\mathcal{N}$. The claim can then be rephrased as $\sum_{v \in V(H)} (\mathrm{d}(v) - 2) = O(\gamma)$, which holds by Lemma 8.2 in Section 8.5. □

(2) The width of (T, μ) is at most $27 \cdot \mathbf{bw}(G) + O(\gamma)$.

For simplicity, let $k = \mathbf{bw}(G)$. By Proposition 8.1, each polyhedral component H is is a minor of G, hence $\mathbf{bw}(H) \leq k$ for all $H \in \mathcal{G}$. In Step 1 of Algorithm 2, we compute a branch decomposition (T'_H, μ'_H) of H of width at most $k' = \frac{9}{2}k$, using Amir's algorithm [35, Theorem 3.8]. In Step 2, we transform (T'_H, μ'_H) to a carving decomposition (T_H^c, μ_H^c) of the medial graph M_H of H of width at most $12k'$, using Lemma 8.4. In Step 3, we transform (T_H^c, μ_H^c) to a *bond* carving decomposition (T_H^b, μ_H^b) of M_H of width at most $12k'$, using the algorithm of [180]. Then, using

Observation 8.2, we transform in Step 4 (T_H^b, μ_H^b) to a branch decomposition (T_H, μ_H) of H. By the proof of Claim 8.1, the discrepancy between $\mathbf{w}(T_H, \mu_H)$ and $\mathbf{w}(T_H^b, \mu_H^b)/2$ is at most the bound provided by Lemma 8.2, i.e., $O(\gamma)$. Therefore, $\mathbf{w}(T_H, \mu_H) \leq 6k' + O(\gamma) = 27k + O(\gamma)$, for all $H \in \mathcal{G}$.

Finally, we merge the branch decompositions of the polyhedral components to obtain a branch decomposition (T, μ) of G, by adding the vertices in A to all the middle sets. Combining the discussion above with Lemmata 8.5 and 8.6, and using that $|A| = O(\gamma)$, we get that

$$\begin{aligned} \mathbf{w}(T, \mu) &\leq \max\{2, \{\mathbf{w}(T_H, \mu_H) \mid H \in \mathcal{G}\}\} + |A| \\ &\leq 27k + O(\gamma) + |A| \\ &= 27k + O(\gamma). \end{aligned}$$

(3) Algorithm 2 runs in $2^{3k+O(\log k)}n^3$ steps.

We analyze sequentially the running time of each step. First, we compute a polyhedral decomposition of G using Algorithm 1 in $O(n^3)$ steps, by Proposition 8.1. Then, we run Amir's algorithm in each component in Step 1, which takes $O(2^{3k}k^{3/2}n^2)$ steps [35, Theorem 3.8]. Step 2 can be done in linear time by Lemma 8.4. Step 3 can be done in $O(n^2)$ time [180]. Step 4 takes linear time by Observation 8.2. Merging the branch decompositions can clearly be done in linear time. Finally, since any two elements in $\mathcal{G}$ share at most two vertices, the overall running time is the claimed one.

□

How surface cut decompositions are used for dynamic programming. We shall now discuss how surface cut decompositions guarantee good upper bounds on the size of the tables of dynamic programming algorithms for problems in Category (C). The size of the tables depends on how many ways a partial solution can intersect a middle set during the dynamic programming algorithm. The interest of a surface cut decomposition is that the middle sets are placed on the surface in such a way that permits to give a precise asymptotic enumeration of the size of the tables. Indeed, in a surface cut decomposition, once we remove a set of vertices whose size is linearly bounded by γ, the middle sets are either of size at most two (in which case the size of the tables is bounded by a constant) or are situated around a set of $O(\gamma)$ nooses, where vertices can be repeated at most $O(\gamma)$ times. In such a setting, the number of ways that a partial solution can intersect a middle set is bounded by the number of non-crossing partitions of the boundary-vertices in a fat-connected subset of the surface (see Definition 8.2). By splitting the boundary-vertices that belong to more than one noose, we can assume that these nooses are mutually disjoint. That way, we reduce the problem to the enumeration of the non-crossing partitions of $O(\gamma)$ disjoint nooses containing ℓ vertices, which are $2^{O(\ell)} \cdot \ell^{O(\gamma)} \cdot \gamma^{O(\gamma)}$, as we prove in the following section (Theorem 8.3). Observe that the splitting operation increases the size of the middle sets by at most $O(\gamma)$, therefore $\ell = k + O(\gamma)$ and this yields an upper bound of $2^{O(k)} \cdot k^{O(\gamma)} \cdot \gamma^{O(\gamma)}$ on the size of the tables of the dynamic programming. In Section 8.7 we use singularity analysis over expressions obtained by the symbolic method to count the number of such non-crossing partitions. Namely, in Sections 8.7.1 and 8.7.2 we give a

precise estimate of the number of non-crossing partitions in surfaces with boundary. Then we incorporate in Section 8.7.3 two particularities of surface cut decompositions: Firstly, we deal with the set A of vertices originating from the polyhedral decomposition. These vertices are not situated around the nooses that disconnect the surface into two connected components, and this is why they are treated as *apices* in the enumeration. Secondly, we take into account that, in fact, we need to count the number of non-crossing *packings* rather than the number of non-crossing partitions, as a solution may not intersect *all* the vertices of a middle set, but only a subset. The combinatorial results of Section 8.7 are of interest by themselves, as they are a natural extension to higher-genus surfaces of the classical non-crossing partitions in the plane, which are enumerated by the Catalan numbers (see e.g. [114]).

8.7 Non-crossing Partitions in Surfaces with Boundary

In this section we obtain upper bounds for non-crossing partitions in surfaces with boundary. The concept of a non-crossing partition in a general surface is not as simple as in the case of the disk, and must be defined carefully. In Section 8.7.1 we set up our notation. In Section 8.7.2 we obtain a tree-like structure that provides a way to obtain asymptotic estimates. As a main tool we employ map enumeration techniques. To conclude, we extend in Section 8.7.3 the enumeration to two more general families.

8.7.1 2-zone decompositions and non-crossing partitions

Let Σ be a surface with boundary. A 2-*zone decomposition of* Σ is a decomposition of Σ where all vertices lay in the boundary of Σ and there is a coloring of the faces using 2 colors (black and white) such that every vertex is incident (possibly more than once) with a unique black face. Black faces are also called *blocks*. A 2-zone decomposition is *regular* if every block is contractible. All 2-zone decompositions are *rooted*: every connected component of the boundary of Σ is edge-rooted. We denote by $\mathcal{S}_\Sigma(k), \mathcal{R}_\Sigma(k)$ the set of general and regular 2-zone decompositions of Σ with k vertices, respectively. A 2-zone decomposition s over Σ defines a non-crossing partition $\pi_\Sigma(s)$ over the set of vertices. Let $\Pi_\Sigma(k)$ be the set of non-crossing partitions of Σ with k vertices. The main objective in this section consists in obtaining bounds for $|\Pi_\Sigma(k)|$. The critical observation is that each non-crossing partition is defined by a 2-zone decomposition. Consequently, $|\Pi_\Sigma(k)| \leq |\mathcal{S}_\Sigma(k)|$. The strategy to enumerate this second set consists in reducing the enumeration to simpler families of 2-zone decompositions. More specifically, the following proposition shows that it is sufficient to study regular decompositions:

Proposition 8.2 *Let $s \in \mathcal{S}_\Sigma$ be a 2-zone decomposition of Σ and let $\pi_\Sigma(s)$ be the associated non-crossing partition. Then there exists a regular 2-zone decomposition $m \in \mathcal{R}_\Sigma$ such that $\pi_\Sigma(s) = \pi_\Sigma(m)$.*

Proof: First we need a definition and a basic topological lemma. Let Σ_1 and Σ_2 be surfaces with boundary, possibly not connected. We write $\Sigma_2 \subset \Sigma_1$ if there exists a continuous injection $i : \Sigma_2 \hookrightarrow \Sigma_1$ such that $i(\Sigma_2)$ is homeomorphic to Σ_2. If s is a 2-zone decomposition

of Σ_2 and $\Sigma_2 \subset \Sigma_1$, then the injection i induces a 2-zone decomposition $i(s)$ on Σ_1 such that $\pi_{\Sigma_2}(s) = \pi_{\Sigma_1}(i(s))$. In other words, all 2-zone decompositions over Σ_2 can be realized on a surface Σ_1 which contains Σ_2. Consequently, informally speaking, $\Pi_{\Sigma_2}(k) \subseteq \Pi_{\Sigma_1}(k)$ if $\Sigma_2 \subset \Sigma_1$.

Lemma 8.9 *Let m^* be a regular 2-zone decomposition of Σ_1. Let $\Sigma_1 \subset \Sigma$. Then m^* defines a regular 2-zone decomposition m over Σ such that $\pi_{\Sigma_1}(m^*) = \pi_\Sigma(m)$.*

Proof: [of Lemma 8.9] Let $i : \Sigma_1 \hookrightarrow \Sigma$ be the corresponding injective application, and consider $m = i(m^*)$. In particular, a block π of m^* is topologically equivalent to the block $i(\pi)$: i is a homeomorphism between Σ and $i(\Sigma)$. Hence $i(\pi)$ is an open contractible set and m is regular. □

The basic idea to prove Proposition 8.2 is the construction of a finite sequence of 2-zone decompositions $s_0 = s, s_1, \ldots, s_t = m$, such that $\pi_\Sigma(s_0) = \cdots = \pi_\Sigma(s_t)$ and $s_t = m$ is regular. First, consider a non-contractible block f of s. Suppose that the boundary of f consists of more than one connected component. We define the operation of *joining boundaries* as follows: let l be a path that joins a vertex v in one component of the boundary of f with a vertex u in another component. This path exists because f is a face. Consider also a pair of paths l_1, l_2 that joins these two vertices around the initial path l, as illustrated in Figure 8.2. We define the face f' as the one obtained from f by deleting the face defined by l_1 and l_2 which contains l. Let s_1 be the resulting 2-zone decomposition. Observe that the number of connected components of the boundary of f' is the same as for f minus one, and that $\pi_\Sigma(s) = \pi_\Sigma(s_1)$. We can apply this argument over f as many times as the number of components of the boundary of f. At the end, we obtain a 2-zone decomposition s_{p_1} such that $\pi_\Sigma(s) = \pi_\Sigma(s_1) = \cdots = \pi_\Sigma(s_{p_1})$.

Suppose now that the boundary of the block f has one connected component. Additionally, in this block there are vertices on its boundary which are incident with f more than once. Let v be a vertex incident $r > 1$ times with f. In this case we define the operation of *cutting vertices* as follows: consider the intersection of a small neighborhood of v with f, and delete vertex v. This intersection has r connected components. We define a face by pasting v with only one of these components, and disconnecting the others from v (see Figure 8.2). Then the resulting 2-zone decomposition has the same associated non-crossing partition, and v is incident with the corresponding block exactly once.

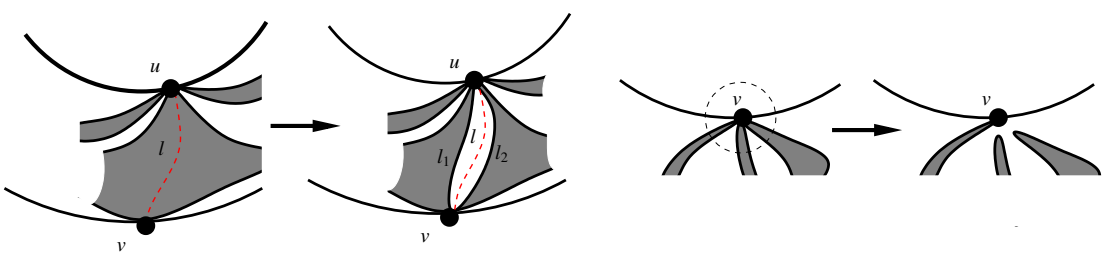

Figure 8.2: The operations of joining boundaries and cutting vertices.

Summarizing, we construct from s a regular 2-zone decomposition in the following way: apply the operation of joining boundaries, and then the operation of cutting vertices. After

this, every block has one boundary and each vertex is incident with its corresponding block exactly once. In this case, a block is either contractible or not. If it is not contractible, let $\mathbb{S}^1$ be a non-contractible cycle, which can be cut using the operator $\succ\!\!\circ$. For all blocks, the number of times we need to apply this operator is bounded by $\gamma(\Sigma)$. At the end, all blocks are contractible and the resulting surface is $\Sigma_1 \subset \Sigma$. So, the resulting 2-zone decomposition m^* is regular, and then by Lemma 8.9 there exists a regular 2-zone decomposition m over Σ such that $\pi_\Sigma(m) = \pi_{\Sigma_1}(m^*) = \pi_\Sigma(s)$, as claimed. □

In other words, $|\Pi_\Sigma(k)| \leq |\mathcal{R}_\Sigma(k)|$ for each value of k. Instead of counting $|\mathcal{R}_\Sigma(k)|$, we reduce our study to the family of regular 2-zone decompositions where each face (block or white face) is contractible. The reason is, as we show later, that this subfamily provides the greatest contribution to the asymptotic enumeration. This set is called the set of *irreducible* 2-zone decompositions of Σ, and it is denoted by $\mathcal{P}_\Sigma(k)$. Equivalently, an irreducible 2-zone decomposition cannot be realized in a proper surface contained in Σ. The details follow.

A generalization of the notion of irreducibility. We shall provide an equivalent definition and an additional property of irreducible 2-zone decompositions. We need the definition stated in the proof of Proposition 8.2 about inclusion of surfaces.

We say that a non-crossing partition π_{Σ_1} is *irreducible* in Σ_1 if there is no realization of π_{Σ_1} in a surface Σ_2 such that $\Sigma_2 \subset \Sigma_1$. This definition is compatible with the notion introduced in Section 8.7.1, as shown in the following lemma:

Lemma 8.10 *Let m be an irreducible 2-zone decomposition of Σ. Then the faces of m are all contractible.*

Proof: We only need to deal with white faces. For a white face whose interior is not an open polygon, there exists a non-contractible cycle $\mathbb{S}^1$. Cutting along $\mathbb{S}^1$ using the operator $\succ\!\!\circ$ we obtain a new surface with boundary Σ' such that $\Sigma' \subset \Sigma$ and m is induced in Σ'. As a conclusion, all faces are contractible. □

8.7.2 Tree-like structures, enumeration, and asymptotic counting

In this subsection we provide estimates for the number of irreducible 2-zone decompositions, which are obtained directly for the surface Σ. This approach is novel and gives upper bounds close to the exact values. The usual technique consists in reducing the enumeration to surfaces of smaller genus, and returning back to the initial one by topological "pasting" arguments. The main point consists in exploiting tree-like structures of the dual graph associated to an irreducible 2-zone decomposition. The main ideas are inspired by [54], where they are used in the context of map enumeration. For simplicity of the presentation, the construction is explained on the disk. The dual graph of a non-crossing partition on the disk is a tree whose internal vertices are bicolored (black color for blocks). An example of this construction is shown in Figure 8.3.

We use this family of trees (and some related ones) in order to obtain a decomposition of elements of the set $\mathcal{P}_\Sigma(k)$. In Section 8.8 the enumeration of this basic family is done,

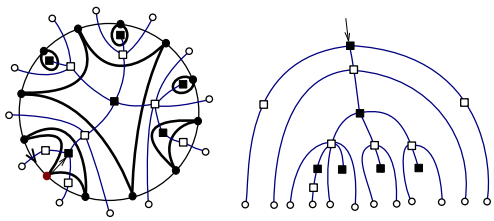

Figure 8.3: A non-crossing partition tree.

as well as the enumeration of the related families. The construction for general surfaces is a generalization of the previous one. An example is shown in the leftmost picture of Figure 8.4. For an element $m \in \mathcal{P}_\Sigma(k)$, denote by M the resulting map on $\overline{\Sigma}$ (recall the definition of $\overline{\Sigma}$ in Section 8.2.1). From M we reconstruct the initial 2-zone decomposition m by pasting vertices of degree 1 which are incident to the same face, and taking the dual map. From M we define a new rooted map on $\overline{\Sigma}$ in the following way: we start deleting recursively vertices of degree 1 which are not roots. Then we continue dissolving vertices of degree 2. The resulting map has $\beta(\Sigma)$ faces and all vertices have degree at least 3 (apart from root vertices, which have degree 1). The resulting map is called the *scheme associated* to M; we denote it by S_M. See Figure 8.4 for an example.

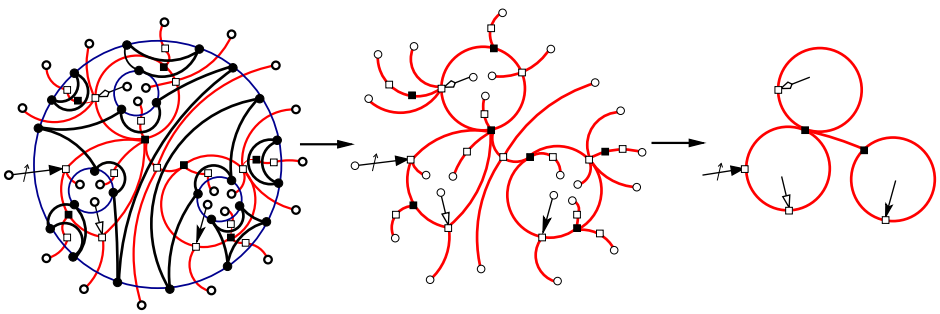

Figure 8.4: The construction of the scheme of an element in $\mathcal{P}_\Sigma$. We consider the dual of an irreducible 2-zone decomposition (leftmost figure). After deleting vertices of degree 1 recursively and dissolving vertices of degree 2, we obtain the associated scheme (rightmost figure).

An inverse construction can be done using maps over $\overline{\Sigma}$ and families of plane trees. Using these basic pieces, we can reconstruct all irreducible 2-zone decompositions. The details of this construction can be found in Section 8.8. Exploiting this decomposition and using singularity analysis (see Section 8.2.4 for the basic definitions), we get the following theorem (Γ denotes the classical Gamma function [115]):

Theorem 8.2 *Let Σ be a surface with boundary. Then the number $|\Pi_\Sigma(k)|$ verifies*

$$|\Pi_\Sigma(k)| \leq_{k\to\infty} \frac{C(\Sigma)}{\Gamma\left(3/2\gamma(\Sigma)+\beta(\Sigma)-3\right)} \cdot k^{3/2\gamma(\Sigma)+\beta(\Sigma)-4} \cdot 4^k, \tag{8.2}$$

where $C(\Sigma)$ is a function depending only on Σ that is bounded by $\gamma(\Sigma)^{O(\gamma(\Sigma))}$.

The steps towards the proof of Theorem 8.2 are included in Sections 8.9, 8.10, and 8.11. Basically, we start characterizing the combinatorial decomposition in terms of plane trees. This combinatorial decomposition is exploited in Proposition 8.3 of Section 8.9 in order to count irreducible 2-zone decompositions. The constant $C(\Sigma)$ is related to the enumeration of cubic maps [41, 131]. Bounds for $C(\Sigma)$ are given in Section 8.10 (see Proposition 8.5). Finally, we prove in Section 8.11 that the asymptotic of $|\mathcal{R}_\Sigma(k)|$ coincides with the one obtained for irreducible 2-zone decompositions. The argument uses a double induction on the number of boundaries and the genus of the surface, and Lemma 8.3 of Section 8.5.

8.7.3 Additional constructions

In the previous section, we enumerated families of non-crossing partitions with boundary. In this section we first deal with a set of additional vertices that play the role of *apices* (cf. the last paragraph of Section 8.6). Secondly, we show how to extend the enumeration from non-crossing partitions to non-crossing packings. In both cases, we show that the modification over generating functions (GFs for short) does not depend on the surface Σ where non-crossing partitions are considered. The analysis consists in symbolic manipulation of GFs and application of singularity analysis over the resulting expressions.

The first problem can be stated in general as follows: for a sequence of positive numbers $\{p_{k,r}\}$, such that we know the GF $\sum_{k,r>0} p_{k,r} z^k u^r$, we want to estimate the value of $\sum_{r=1}^{k} r^l p_{k,r}$ for a fixed value l. This problem arises from the fact that we have a set of vertices (the *apices*), such that every vertex of the set can be associated to an arbitrary block of a non-crossing partition. The details of the analysis of this problem are done in Section 8.12. Basically, this problem only introduces a variation in the subexponential term of the asymptotic stated in Theorem 8.2. The second problem consists in generalizing from non-crossing partitions to non-crossing packings. In other words, for a fixed number of k vertices, fix an arbitrary subset of $i \leq k$ vertices, and consider the set of non-crossing partitions on Σ on this set of i vertices. This value is precisely $|\Pi_\Sigma(i)|$. Among the total set of k vertices, this set of i vertices can be chosen in $\binom{k}{i}$ ways. Hence, we want to estimate the sum $\sum_{i=0}^{k} \binom{k}{i} |\Pi_\Sigma(i)|$. Observe that this construction is quite close to Bell numbers, which count the number of ways a set of k elements can be partitioned into nonempty subsets. The details of the analysis can be found in Section 8.13. In this case, a combinatorial trick (Lemma 8.12) shows that the modification only affects the base of the exponential term.

Combining the univariate asymptotic obtained in Theorem 8.2 with the constructions described above (the details can be found in Propositions 8.7 and 8.8 in Sections 8.12 and 8.13, respectively) we obtain the following theorem, which gives the bound on the size of the tables when using surface cut decompositions:

Theorem 8.3 *Let $\overline{\Pi_{\Sigma,l}}(k)$ be the set of non-crossing partitions of Σ with k vertices and a set of l apices. Then the value $\sum_{i=0}^{k} \binom{k}{i} \left|\overline{\Pi_{\Sigma,l}}(k)\right|$ is upper-bounded, for large k, by*

$$\frac{C(\Sigma)}{2^{2+l}\Gamma\left(3/2\gamma(\Sigma)+\beta(\Sigma)-3\right)} \cdot k^{3/2\gamma(\Sigma)+\beta(\Sigma)-4+l} \cdot 5^{k+1}, \tag{8.3}$$

where $C(\Sigma)$ is a function depending only on Σ that is bounded by $\gamma(\Sigma)^{O(\gamma(\Sigma))}$.

8.8 Enumeration of Non-crossing Partitions of the Disk and Related Constructions

In this section we introduce some terminology related to trees that arise as dual graphs of non-crossing partitions on the disk. Then, we use these concepts to obtain the number of non-crossing partitions of the disk with n vertices. At last, we introduce some families of related trees, which are used in the construction of the dual map of a non-crossing partition in a surface of higher genus.

The dual graph of a non-crossing partition is a tree, which is called the (non-crossing partition) tree associated to the non-crossing partition. Vertices of degree 1 (that is, the leafs of the tree) are called *danglings*. Vertices of the tree are called *block vertices* if they are associated to a block of the non-crossing partition. The remaining vertices are either *non-polygon vertices* or *danglings*. By construction, all vertices adjacent to a polygon vertex are non-polygon vertices. Conversely, each vertex adjacent to a non-polygon vertex is either a block vertex or a dangling. Graphically, we use the symbols ■ for block vertices, □ for non-polygon vertices and ∘ for danglings.

Denote by $\mathcal{T}$ the set of non-crossing partitions trees, and let $\mathbf{T} = \mathbf{T}(z,u) = \sum_{k,n>0} \mathbf{t}_{k,m} z^k u^m$ be the corresponding GF. The variable z marks danglings and u marks block vertices. We use also an auxiliary family $\mathcal{B}$, defined as the set of trees which are rooted at a non-polygon vertex. Let $\mathbf{B} = \mathbf{B}(z,u) = \sum_{k,n>0} \mathbf{b}_{k,m} z^k u^m$ be the associated GF. The next lemma gives the exact enumeration of $\mathcal{T}$ and $\mathcal{B}$.

Lemma 8.11 *The number of non-crossing trees counted by the number of danglings and block vertices is enumerated by*

$$\mathbf{T}(z,u) = \frac{1 - z(1-u) - \sqrt{(z(1-u)-1)^2 - 4zu}}{2zu}. \tag{8.4}$$

Furthermore, $\mathbf{B}(z,u) = z\mathbf{T}(z,u)$.

Proof: We establish combinatorial relations between $\mathcal{B}$ and $\mathcal{T}$, from which we deduce the desired result. First, observe that there is no restriction on the size of the blocks. Hence the degree of every block vertex is arbitrary. This condition is translated symbolically via the following relation: $\mathcal{T} = \blacksquare \times \mathrm{Seq}(\mathcal{B})$. Similarly, $\mathcal{B}$ can be written in the form $\mathcal{B} = \{\circ\} \times \mathrm{Seq}(\mathcal{T} \times \{\circ\})$.

These combinatorial conditions translate using Table 8.1 into the system of equations

$$\mathbf{T} = \frac{u}{1-\mathbf{B}}, \ \mathbf{B} = \frac{z}{1-z\mathbf{T}}.$$

If we substitute the expression of $\mathbf{B}$ in the first equation, one obtains that $\mathbf{T}$ satisfies the equation $z\mathbf{T}^2 + (z(1-u)-1)\mathbf{T} + u = 0$. The valid solution of this equation is (8.4). Solving the previous system of equations in terms of $\mathbf{B}$, we obtain that $\mathbf{B} = z\mathbf{T}$, as claimed. □

Observe that writing $u = 1$, we obtain that $\mathbf{T}(z) = \mathbf{T}(z,1) = \frac{1-\sqrt{1-4z}}{2z}$, and $\mathbf{B}(z) = \mathbf{B}(z,1) = z\mathbf{T}(z)$, and we recover the GF for Catalan numbers.

We need also a set of families of trees that are quite related to the previous ones. We call them *double trees*. A double tree is obtained in the following way: consider a path where we concatenate vertices of type ■ with vertices of type □. A double tree is a tree obtained by pasting on every internal vertex of type ■ a pair of elements of $\mathcal{T}$ (one at each side of the path), and similarly for internal vertices of type □. We say that a double tree is of type either ■ − ■ ■ − □, or □ − □ depending on the type of the ends of the path. An example for a double tree of type ■ − ■ is shown in Figure 8.5.

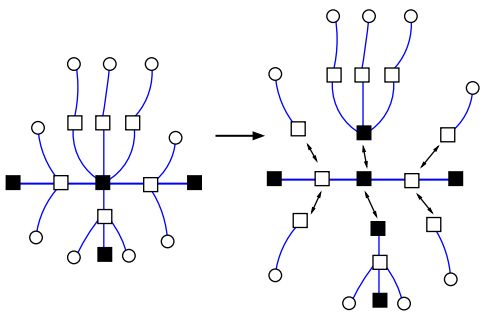

Figure 8.5: A double tree and its decomposition.

We denote these families by $\mathcal{T}_{■-■}$, $\mathcal{T}_{□-■}$, and $\mathcal{T}_{□-□}$, and the corresponding GF by $\mathbf{T}_1(z,u) = \mathbf{T}_1$, $\mathbf{T}_2(z,u) = \mathbf{T}_2$ and $\mathbf{T}_3(z,u) = \mathbf{T}_3$, respectively. Recall that in all cases z marks danglings and u marks block vertices. A direct application of the symbolic method provides a way to obtain explicit expressions for the previous GFs. The decomposition and the GFs of the three families is summarized in Table 8.2.

Family	Developement	Compact expression
$\mathcal{T}_{□-■}$	$1 + \frac{1}{u}\mathbf{B}^2\mathbf{T}^2 + \frac{1}{u^2}\mathbf{B}^4\mathbf{T}^4 + \dots$	$1/(1 - \mathbf{T}^2\mathbf{B}^2/u)$
$\mathcal{T}_{■-■}$	$\mathbf{B}^2 + \frac{1}{u}\mathbf{B}^4\mathbf{T}^2 + \frac{1}{u^2}\mathbf{B}^6\mathbf{T}^4 + \dots$	$\mathbf{B}^2/(1 - \mathbf{T}^2\mathbf{B}^2/u)$
$\mathcal{T}_{□-□}$	$\frac{1}{u}\mathbf{T}^2 + \frac{1}{u^2}\mathbf{B}^2\mathbf{T}^4 + \frac{1}{u^3}\mathbf{B}^4\mathbf{T}^6 + \dots$	$\frac{1}{u}\mathbf{T}^2/(1 - \mathbf{T}^2\mathbf{B}^2/u)$

Table 8.2: GFs for double trees.

To conclude, the family of *pointed* non-crossing trees $\mathcal{T}^\bullet$ is built pointing a dangling over each tree. In this case, the associated GF is $\mathbf{T}^\bullet = z\frac{\partial}{\partial z}\mathbf{T}$. Similar definitions can be done for the family $\mathcal{B}$. Pointing a dangling define a unique path between this distinguished dangling and the root of the tree.

8.9 Combinatorial Decomposition and Enumeration

Throughout this section, we use the notation and definitions introduced in Section 8.8 (i.e., families of trees, double trees and pointed trees, and the corresponding GFs). To simplify the notation, we denote by $\mathcal{P}_\Sigma(k,m)$ the set of irreducible 2-zone decompositions of Σ with

k vertices and m blocks. We write $p^{\Sigma}_{k,m}$ for the cardinal of this set. Let $p^{\Sigma}_{k} = \sum_{m>0} p^{\Sigma}_{k,m}$. The GF associated to the numbers $p^{\Sigma}_{k,m}$ is denoted by $\mathbf{P}_{\Sigma}(z,u)$. We denote by $\mathfrak{S}_{\Sigma}$ the set of rooted maps on $\overline{\Sigma}$ with $\beta(\Sigma)$ faces, whose vertices are bicolored (either ■ or □) and have degree at least 3. In particular, endpoints of a given edge can have the same color. This notation is used in Sections 8.11, 8.12, and 8.13. Observe that in our framework, each map has $\beta(\Sigma)$ roots, in contrast to the classical theory of enumeration of rooted maps (where a unique root is considered).

Applying Euler's formula for maps on $\overline{\Sigma}$ imples that $|\mathfrak{S}_{\Sigma}|$ is finite, because the number of faces is fixed. It is also obvious that if S_M is the scheme associated to a map M, then $S_M \in \mathfrak{S}_{\Sigma}$. These observations provide a way to establish a combinatorial bijection, that can be exploited to obtain the enumeration of $\mathcal{P}_{\Sigma}$. More concretely, each element M can be constructed from an element S of $\mathfrak{S}_{\Sigma}$ in the following way:

1. For an edge of S with both end-vertices of type ■, we paste a double tree of type ■ − ■ along it. Similar operations can be realized for edges with end-vertices $\{□, ■\}$ and $\{□, □\}$.
2. For a block vertex v of S, not incident with any root, we paste $\mathrm{d}(v)$ elements of $\mathcal{T}$ (identifying the roots of trees in $\mathcal{T}$ with v), one in each region determined by consecutive half-edges.
3. For a set of roots with an end-point in the same block vertex v, we paste an element of $\mathcal{T}^{\bullet}$ along each one of the roots (the marked leaf determines which is the dangling root). Over v we paste trees of $\mathcal{T}$ as we have done in the previous case. We do not paste trees between a root and a half-edge of S. A similar operation is done if the vertex is of type □.

This construction is shown for a concrete example in Figure 8.6.

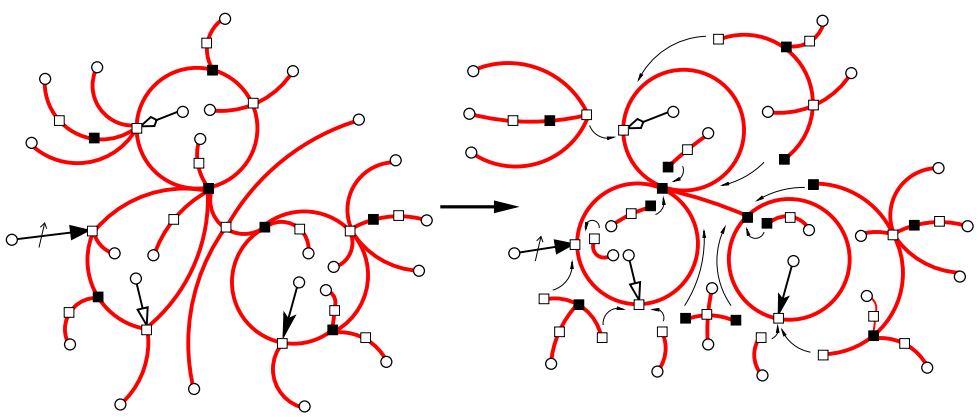

Figure 8.6: The decomposition into bicolored trees and the associated scheme.

Let us introduce some notation. Consider an element S of $\mathfrak{S}_{\Sigma}$. Let $v_1(S), v_2(S)$ be the set of block vertices and non-polygon vertices of S, respectively. Write $B(S), W(S)$ for the number of roots which are incident with a vertex of type ■ and □, respectively. In particular, $B(S) + W(S) = \beta(\Sigma)$. Denote by $e_1(S)$ the number of edges in S of type ■ − ■.

We similarly define $e_2(S)$ and $e_3(S)$ for edges of type $\square - \blacksquare$ and $\square - \square$, respectively. Observe that $e_1(S) + e_2(S) + e_3(S) + B(S) + W(S)$ is the number of edges of S, that is $e(S) = |E(S)|$. For a vertex x of S, denote by $r(x)$ the number of roots which are incident with it.

The previous decomposition provides a direct way to obtain the desired enumeration.

Proposition 8.3 *Let Σ be a surface with boundary. Then the coefficient $[z^k]\mathrm{P}_\Sigma(z,1)$ has an asymptotic expansion of the form*

$$[z^k]\mathrm{P}_\Sigma(z,1) = p_k^\Sigma =_{k\to\infty} \frac{C(\Sigma)}{\Gamma(-3\chi(\Sigma)/2 + \beta(\Sigma))} k^{-3\chi(\Sigma)/2+\beta(\Sigma)-1} 4^k (1 + O(k^{-1/2})), \tag{8.5}$$

where $C(\Sigma)$ is a function depending only on Σ.

Proof: According to the previous observations, $\mathrm{P}_\Sigma(z,u)$ can be written in the following form: for each $S \in \mathfrak{S}_\Sigma$, we replace edges (not roots) with double trees, roots with pointed trees, and vertices with sets of trees. More concretely, the GF we obtain is

$$\sum_{S\in\mathfrak{S}_\Sigma} u^{|v_1(S)|}\mathbf{T}_1^{e_1(S)}\mathbf{T}_2^{e_2(S)}\mathbf{T}_3^{e_3(S)}\left(\frac{\mathbf{T}}{u}\right)^{\sum_{x\in v_1(S)}(\mathrm{d}(x)-2r(x))}\mathbf{B}^{\sum_{y\in v_2(S)}(d(y)-2r(y))}\left(\frac{\mathbf{T}^\bullet}{u}\right)^{B(S)}(\mathbf{B}^\bullet)^{W(S)}. \tag{8.6}$$

Observe that terms $\mathbf{T}$ and $\mathbf{T}^\bullet$ appear divided by u. The reason is that we paste non-crossing trees through the root, which is a block vertex. In order to do it, we delete the corresponding block vertex, we paste the trees identifying their roots without counting the root, and finally we add the total number of block vertices (thus the term $u^{|v_1(S)|}$). To obtain the asymptotic behavior in terms of the number of danglings, we write $u = 1$ in Equation (8.6). To study the resulting GF, we need the expression of each factor of Equation (8.6) when we write $u = 1$. In Table 8.3 all the expressions are shown. This table is built from the expressions for $\mathbf{T}$ and $\mathbf{B}$ deduced in Lemma 8.11 and the expressions for double trees in Table 8.2. The GF in Equation (8.6) is a finite sum (a total of $|\mathfrak{S}_\Sigma|$

GF	Expression
$\mathbf{T}_1(z)$	$1/16(1-4z)^{-1/2} - 1/8(1-4z)^{1/2} + 1/16(1-4z)^{3/2}$
$\mathbf{T}_2(z)$	$1/4(1-4z)^{-1/2} + 1/2 + (1-4z)^{1/2}$
$\mathbf{T}_3(z)$	$z^2\left(1/16(1-4z)^{-1/2} - 1/8(1-4z)^{1/2} + 1/16(1-4z)^{3/2}\right)$
$\mathbf{T}(z)$	$1/(2z)(1-(1-4z)^{1/2})$
$\mathbf{B}(z)$	$1/2\left(1-(1-4z)^{1/2}\right)$
$\mathbf{T}^\bullet(z)$	$1/z(1-4z)^{-1/2} - 1/(2z^2)(1-(1-4z)^{-1/2})$
$\mathbf{B}^\bullet(z)$	$(1-4z)^{-1/2}$

Table 8.3: Univariate GF for all families of trees.

terms), so its singularity is located at $z = 1/4$ (since each addend has a singularity at this point). For each choice of S,

$$\mathbf{T}(z,1)^{\sum_{x\in v_1(S)}(\mathrm{d}(x)-2r(x))}\mathbf{B}(z,1)^{\sum_{y\in v_2(S)}(\mathrm{d}(y)-2r(y))} = \sum_{n=0}^{f(S)} f_n(z)(1-4z)^{n/2}, \tag{8.7}$$

where the positive integer $f(S)$ depends only on S, $f_n(z)$ are functions analytic at $z = 1/4$, and $f_0(z) \neq 0$ at $z = 1/4$. For the other multiplicative terms, we obtain

$$\mathbf{T}_1(z,1)^{e_1(S)}\mathbf{T}_2(z,1)^{e_2(S)}\mathbf{T}_3(z,1)^{e_3(S)}\mathbf{T}^\bullet(z,1)^{B(S)}\mathbf{B}^\bullet(z,1)^{W(S)} = G_S(z)(1-4z)^{-\frac{e(S)}{2}} + \dots, \quad (8.8)$$

where $G_S(z)$ is an analytic function at $z = 1/4$. The reason of this fact is that each GF in the previous formula can be written in the form $p(z)(1-4z)^{-1/2} + \dots$, where $p(z)$ is a function analytic at $z = 1/4$, and $e_1(S) + e_2(S) + e_3(S) + B(S) + W(S)$ is the total number of edges. Multiplying Equation (8.7) and Expression (8.8) we recover the contribution of a map S in $\mathrm{P}_\Sigma(z,1)$. More concretely, the contribution of a single map S to Equation (8.6) can be written in the form

$$g_S(z)(1-4z)^{-e(S)/2} + \dots,$$

where $g_S(z)$ is an analytic function at $z = 1/4$. From Equation (8.1), the maps giving the greatest contribution to the asymptotic of p_k^Σ are the ones which maximize the value of $e(S)$. Applying Euler's formula (recall that all maps in $\mathfrak{S}_\Sigma$ have $\beta(\Sigma)$ faces) on $\overline{\Sigma}$ gives that these maps are the ones where each vertex have degree 3 (i.e., cubic maps). In particular, cubic maps with $\beta(\Sigma)$ faces and $\beta(\Sigma)$ roots have $2\beta(\Sigma) - 3\chi(\Sigma)$ edges. Hence, as a consequence of the Transfer Theorem for singularity analysis, the singular expansion of $\mathrm{P}_\Sigma(z,1)$ at $z = 1/4$ is

$$\mathrm{P}_\Sigma(z,1) =_{z\to 1/4} C(\Sigma)(1-4z)^{3\chi(\Sigma)/2-\beta(\Sigma)}\left(1 + O((1-4z)^{1/2})\right), \quad (8.9)$$

where $C(\Sigma) = \sum_{S\in\mathfrak{S}_\Sigma} g_S(1/4)$. Applying the Transfer Theorem in this expression yields the claimed result. □

8.10 Bounding $C(\Sigma)$ in Terms of Cubic Maps

In this section we obtain bounds for $C(\Sigma)$. We use the same notation as in Section 8.9. A more refined analysis over functions $g_S(z)$ provides upper bounds for $C(\Sigma)$. This is done in the following proposition:

Proposition 8.4 *The function $C(\Sigma)$ defined in Proposition 8.3 satisfies*

$$C(\Sigma) \leq 2^{\beta(\Sigma)}|\mathfrak{S}_\Sigma|. \quad (8.10)$$

Proof: For each $S \in \mathfrak{S}_\Sigma$, we obtain bounds for $g_S(1/4)$. We use Table 8.4, which is a simplification of Table 8.3. We are only concerned now about the constant term of each GF. Table 8.4 brings the following information: the greatest contribution from double trees, trees, and families of pointed trees comes from $\mathcal{T}_{\square-\blacksquare}$, $\mathcal{T}$, and $\mathcal{T}^\bullet$, respectively. The constants are 1/4, 2, and 4, respectively. Each cubic map has $2\beta(\Sigma) - 3\chi(\Sigma)$ edges ($\beta(\Sigma)$ of them being roots) and $\beta(\Sigma) - 2\chi(\Sigma)$ vertices ($\beta(\Sigma)$ of them being incident with roots). This characterization provides the following upper bound for $g_S(1/4)$:

$$g_S(1/4) \leq \left(\frac{1}{4}\right)^{2\beta(\Sigma)-3\chi(\Sigma)-\beta(\Sigma)} 2^{-3\cdot 2\chi(\Sigma)+\beta(\Sigma)} 4^{\beta(\Sigma)} = 2^{\beta(\Sigma)}. \quad (8.11)$$

□

GF	Expression	Developement at $z = 1/4$
$\mathbf{T}_1(z)$	$1/16(1-4z)^{-1/2} + \dots$	$1/16(1-4z)^{-1/2} + \dots$
$\mathbf{T}_2(z)$	$1/4(1-4z)^{-1/2} + \dots$	$1/4(1-4z)^{-1/2} + \dots$
$\mathbf{T}_3(z)$	$z^2/16(1-4z)^{-1/2} + \dots$	$1/256(1-4z)^{-1/2} + \dots$
$\mathbf{T}(z)$	$1/(2z) + \dots$	$2 + \dots$
$\mathbf{B}(z)$	$1/2 + \dots$	$1/2 + \dots$
$\mathbf{T}^\bullet(z)$	$1/z(1-4z)^{-1/2} + \dots$	$4(1-4z)^{-1/2} + \dots$
$\mathbf{B}^\bullet(z)$	$(1-4z)^{-1/2}$	$(1-4z)^{-1/2}$

Table 8.4: A simplification of Table 8.3 used in Proposition 8.3.

The value of $\Im_\Sigma$ can be bounded using the results in [41, 131]. Indeed, Gao shows in [131] that the number of rooted cubic maps with n vertices in an orientable surface of genus[6] g is asymptotically equal to

$$t_g \cdot n^{5(g-1)/2} \cdot (12\sqrt{3})^n,$$

where the constant t_g tends to 0 as g tends to ∞ [41]. A similar result is also stated in [131] for non-orientable surfaces. By duality, the number of rooted cubic maps in a surface $\overline{\Sigma}$ of genus $\chi(\Sigma)$ with $\beta(\Sigma)$ faces is asymptotically equal to $t_{\chi(\Sigma)} \cdot \beta(\Sigma)^{5(\chi(\Sigma)-1)/2} \cdot (12\sqrt{3})^{\beta(\Sigma)}$. This value is clearly bounded by $\gamma(\Sigma)^{O(\gamma(\Sigma))}$.

To conclude, we observe that the elements of $\Im_\Sigma$ are obtained from rooted cubic maps with $\beta(\Sigma)$ faces by adding a root on each face different from the root face. Observe that each edge is incident with at most two faces, and that the total number of edges is $-3\chi(\Sigma)$. Consequently, the number of ways of rooting a cubic map with $\beta(\Sigma) - 1$ unrooted faces is bounded by $\binom{-6\chi(\Sigma)}{\beta(\Sigma)-1}$, which is bounded by $\gamma(\Sigma)^{O(\gamma(\Sigma))}$.

By the above discussion, the following proposition holds.

Proposition 8.5 *The constant $C(\Sigma)$ verifies*

$$C(\Sigma) \leq t_{\chi(\Sigma)} \cdot \beta(\Sigma)^{5(\chi(\Sigma)-1)/2} \cdot (12\sqrt{3})^{\beta(\Sigma)} \binom{-6\chi(\Sigma)}{\beta(\Sigma)-1} 2^{\beta(\Sigma)}.$$

In particular, $C(\Sigma) = \gamma(\Sigma)^{O(\gamma(\Sigma))}$.

Combining Propositions 8.3 and 8.5, we obtain that

$$p_k^\Sigma \leq_{k\to\infty} \frac{C(\Sigma)}{\Gamma(3/2\gamma(\Sigma) + \beta(\Sigma) - 3)} \cdot k^{3/2\gamma(\Sigma)+\beta(\Sigma)-4} \cdot 4^k, \tag{8.12}$$

where $C(\Sigma) = \gamma(\Sigma)^{O(\gamma(\Sigma))}$ is a function depending only on Σ.

[6] the *genus* $g(\Sigma)$ of an orientable surface Σ is defined as $g(\Sigma) = \gamma(\Sigma)/2$ (see [163]).

8.11 Reducibility vs Irreducibility

In this section we use the same notation as in Section 8.8. For conciseness we use the notation $a(\Sigma)$ to denote the constant term which appears in all the asymptotic expressions in Section 8.8.

For a non-irreducible regular element s of $\mathcal{R}_\Sigma$ (recall Lemma 8.10) there is a non-contractible cycle $\mathbb{S}^1$ contained in a white 2-dimensional region of s. Additionally, s induces a regular 2-zone decomposition over the surface $\Sigma \nearrow \mathbb{S}^1 = \Sigma'$, which can be irreducible or not. By Lemma 8.9, each element of $\mathcal{R}_{\Sigma'}$ defines an element on $\mathcal{R}_\Sigma$. To prove that irreducible 2-zone decompositions over Σ give the maximal contribution to the asymptotic, we apply a double induction argument on the pair $(\gamma(\Sigma), \beta(\Sigma))$. The critical point is the initial step, which corresponds to $\gamma(\Sigma) = 0$:

Proposition 8.6 *Let Σ be a surface obtained from the sphere deleting β disjoints disks. Then*

$$|\mathcal{R}_\Sigma(k)| \leq_{k\to\infty} |\mathcal{P}_\Sigma(k)|.$$

Proof: Induction on β. The case $\beta = 1$ corresponds to the disk. We deduced in Section 8.8 the exact expression for $\mathrm{P}_\Sigma(z,u)$ (see Equation (8.4)). In this case the equality $|\mathcal{R}_\Sigma(k)| = |\mathcal{P}_\Sigma(k)|$ holds for every value of k. Let us consider now the case $\beta = 2$, which corresponds to the cylinder. From Equation (8.12), the number of irreducible 2-zone decompositions over the cylinder verifies

$$|\mathcal{P}_\Sigma(k)| =_{k\to\infty} a(\Sigma) \cdot k \cdot 4^k (1 + O(k^{-1/2})). \tag{8.13}$$

Let us calculate upper bounds for the asymptotic of non-irreducible 2-zone decompositions on a cylinder. A non-contractible cycle $\mathbb{S}^1$ on a cylinder separates it into a pair of cylinders. In other words $\Sigma' = \Sigma \nearrow \mathbb{S}^1$ is a pair of disks. The asymptotic in this case is of the form $[z^k]\mathbf{T}(z,1)^2 =_{k\to\infty} O(k^{-3/2}4^k)$. The subexponential term in Equation (8.13) is greater, so the claim of the proposition holds for $\beta = 1$.

Let us proceed to apply the inductive step. Let $\beta > 1$ be the number of boundaries of Σ. A non-contractible cycle $\mathbb{S}^1$ always separates Σ into two connected components, namely Σ_1 and Σ_2. Let $\beta_1, \beta_2 < \beta$ be the number of boundaries of Σ_1 and Σ_2, respectively. By induction hypothesis,

$$|\mathcal{R}_{\Sigma_j}(k)| \leq_{k\to\infty} |\mathcal{P}_{\Sigma_j}(k)|,$$

for $j = 1, 2$. Consequently, we only need to deal with irreducible decompositions of Σ_1 and Σ_2. The GF of 2-zone regular decompositions that reduces to decompositions over Σ_1 and Σ_2 has the same asymptotic as $\mathrm{P}_{\Sigma_1}(z,1) \cdot \mathrm{P}_{\Sigma_2}(z,1)$. The estimate of its coefficients is

$$[z^k]\mathrm{P}_{\Sigma_1}(z,1) \cdot \mathrm{P}_{\Sigma_2}(z,1) \leq a(\Sigma_1) \cdot a(\Sigma_2)[z^k](1-4z)^{-5/2\beta_1+3} \cdot (1-4z)^{-5/2\beta_2+3} =_{k\to\infty} O\left(k^{5/2\beta-7} \cdot 4^k\right).$$

Consequently, the above term is smaller than p_k^Σ when k is large enough (the value is of the form $\left(k^{5/2\beta-4}4^k\right)$, and does not depend on how Σ is cut. □

The next step is to adapt the previous argument to surfaces of genus greater than 0. Let Σ be a surface with boundary and Euler genus $\gamma(\Sigma)$. Consider a non-contractible cycle $\mathbb{S}^1$ and the resulting surface $\Upsilon = \Sigma \nearrow \mathbb{S}^1$. Two situations can occur:

1. Υ is connected and $\beta(\Upsilon) = \beta(\Sigma)$. In this case, the Euler genus has been decreased by either 1 if the cycle is one-sided or by 2 if the cycle is two-sided. This result appears as Lemma 4.2.4 in [163].
2. The resulting surface is not connected, $\Upsilon = \Upsilon_1 \sqcup \Upsilon_2$. In this case, the total number of boundaries is $\beta(\Upsilon) = \beta(\Upsilon_1) + \beta(\Upsilon_2)$. By Lemma 8.3, $\gamma(\Sigma) = \gamma(\Upsilon_1) + \gamma(\Upsilon_2) - 2$.

The induction argument distinguishes between these two cases: if $\Upsilon = \Sigma \rightarrowtail \mathbb{S}^1$ is connected, by induction on the genus, $|\mathcal{R}_\Upsilon(k)| \leq_{k\to\infty} |\mathcal{P}_\Upsilon(k)|$. Additionally, by Expression (8.12), an upper bound for $|\mathcal{P}_\Upsilon(k)|$ is

$$[z^k]\mathrm{P}_\Upsilon(z,1) = a(\Upsilon)\cdot k^{3/2\gamma(\Upsilon)+\beta(\Upsilon)-4}\cdot 4^k(1+O(k^{-1/2})) =_{k\to\infty} O\left(k^{3/2\gamma(\Sigma)+\beta(\Sigma)-4}\cdot 4^k\right).$$

If Υ is not connected, then $\Upsilon = \Upsilon_1 \sqcup \Upsilon_2$, $\beta(\Sigma) = \beta(\Upsilon) - 2 = \beta(\Upsilon_1) + \beta(\Upsilon_2)$, and $\gamma(\Sigma) = \gamma(\Upsilon_1) + \gamma(\Upsilon_2)$. Again, by induction hypothesis we only need to look at the irreducible ones. Consequently,

$$[z^k]\mathrm{P}_{\Upsilon_1}(z,1)\mathrm{P}_{\Upsilon_2}(z,1) = a(\Upsilon_1)\cdot a(\Upsilon_2)[z^k](1-4z)^{-3/2(\gamma(\Upsilon_1)+\gamma(\Upsilon_2))-(\beta(\Upsilon_1)+\beta(\Upsilon_2))+6}.$$

The exponent of $(1-4z)$ can be written as $-3/2\gamma(\Sigma) - \beta(\Sigma) + 6$. Consequently, the value $[z^k]\mathrm{P}_{\Upsilon_1}(z,1)\mathrm{P}_{\Upsilon_2}(z,1)$ is bounded, for k large enough, by

$$k^{3/2\gamma(\Sigma)+\beta(\Sigma)-6-1}\cdot 4^k = k^{3/2\gamma(\Sigma)+\beta(\Sigma)-7}\cdot 4^k = O\left(k^{3/2\gamma(\Sigma)+\beta(\Sigma)-4}\cdot 4^k\right).$$

Hence the contribution is smaller than the one given by $|\mathcal{P}_\Sigma(k)|$, as claimed.

8.12 Dealing with a Set of Apices

Due to the definition of surface cut decomposition, we need to modify the family $\mathcal{P}_\Sigma$ of irreducible 2-zone decompositions in the following way: consider a set of l vertices $\{\overline{1}, \overline{2}, \ldots, \overline{l}\} = [\bar{l}]$ disjoint from the set of vertices over Σ. This set of vertices is called set of *apices*. For every value of k we want to count the number of pairs (s_r, f), where $s_r \in \mathcal{P}_\Sigma(k)$ has r blocks, and f is an arbitrary application $f : [\bar{l}] \to [r]$. The number of such pairs is $\sum_{r=1}^{k} r^l p_{k,r}^\Sigma$ (recall that the number of irreducible 2-zone decompositions with k vertices and r blocks is $p_{k,r}^\Sigma$, and the associated GF is $\mathrm{P}_\Sigma(z,u)$). The aim of this section is to obtain estimates for this sum. This problem can be stated in the following equivalent way: let $\mathbf{F}(z,u)$ be a GF with expansion $\mathbf{F}(z,u) = \sum_{k,r\geq 0} f_{k,r} z^k u^r$, such that $[z^k u^r]\mathbf{F}(z,u) = 0$ if $r > k$. For a non-negative integer l, we want to estimate the sum

$$\sum_{r=0}^{k} r^l f_{k,r} = [z^k] \sum_{k\geq 0} \sum_{r=0}^{\infty} r^l f_{k,r} z^k u^r \Big|_{u=1},$$

for k large enough. Let Θ be the pointing operator on the second variable: $\Theta\mathbf{F}(z,u) = u\frac{\partial}{\partial u}\mathbf{F}(z,u) = u\mathbf{F}_u(z,u)$. Applying l times the operator Θ over $\mathbf{F}(z,u)$ gives $\Theta^l\mathbf{F}(z,u) = \sum_{k,m\geq 0}^{\infty} m^l f_{k,r} z^k u^m$, so our problem consists in estimating $[z^k]\Theta^l\mathbf{F}(z,u)\big|_{u=1}$. The strategy we use to obtain the estimate consists in simplifying this expression up to a function from which we know to get the asymptotic. Firstly, observe that Θ^l can be written as $\sum_{i=1}^{l} q_i(u)\frac{\partial^i}{\partial u^i}$, where $q_i(u)$ is a polynomial on u. For $i = l$, the value of $q_l(u)$ is u^l. We

show that the greatest contribution to the enumeration comes from the term $i = l$. As a consequence, we only need to deal with $u^l \frac{\partial^l}{\partial u^l}\mathbf{F}(z,u)$. To do this, it is also convenient to observe the following: for $b < a$ positive real numbers, the asymptotic of $(1-4z)^{-a}$ is greater than the one for $(1-4z)^{-b}$: from the Transfer Theorem for singularity analysis (Equation (8.1)), both GFs have an exponential growth of the form 4^k. However, their asymptotic growth is not the same: while the first one has a subexponential growth of the form k^{a-1}, the second one is of the form k^{b-1}, which is smaller. Generalizing this to a linear combination of terms of the form $(1-4z)^{-a_i}$, where a_i is a positive real number, the asymptotic of the whole function comes from the value a_i with the greatest modulus.

Let us return to study $\mathrm{P}_\Sigma(u,z)$. Observe that GFs for double trees can be factorized in the following way (consult Table 8.2):

$$\frac{\mathbf{G}(z,u)}{1-\frac{1}{u}\mathbf{T}^2\mathbf{B}^2/u} = \frac{-2uz^2\mathbf{G}(z,u)}{((u+1)\,z-1)\,\sqrt{(z(1-u)-1)^2-4zu}+z^2(u-1)^2-2z(u+1)+1},$$

where $\mathbf{G}(z,u)$ is either 1, $\mathbf{T}^2/u$ or $\mathbf{B}^2$. For conciseness, write $\mathbf{f} = (z(1-u)-1)^2-4zu$, $\mathbf{g} = ((u+1)\,z-1)$ and $\mathbf{h} = z^2(u-1)^2-2z(u+1)+1$. The previous formula can be written in the form $-2uz^2\mathbf{G}(z,u)/(\mathbf{h}+\mathbf{g}\sqrt{\mathbf{f}})$. Additionally, $\mathbf{g}(z,1) = 2z-1$, $\mathbf{f}(z,1) = 1-4z$ and $\mathbf{h}(z,1) = 1-4z$. For $u = 1$, the smallest singularity of the function is located at $z = 1/4$, where function $\sqrt{\mathbf{f}}$ ceases to be analytic. Consequently, the source of the singularity on a double tree comes exclusively from the term $\sqrt{\mathbf{f}}$. Furthermore, when we write $u = 1$, the smallest singularity of every derivative (with respect to u) of $-2uz^2\mathbf{G}(z,u)/(\mathbf{h}+\mathbf{g}\sqrt{\mathbf{f}})$ is located at $z = 1/4$, because the denominator is always the same (possibly with a greater exponent). A similar argument applies to the families of pointed trees (same behavior, $\mathbf{f}^{-1/2}$).

Taking into account this, and rationalizing the previous expressions, Expression (8.6) can be written in the form

$$\mathrm{P}_\Sigma(z,u) = \sum_{S\in\mathfrak{S}_\Sigma}\left(\mathbf{g}_S(z,u)\mathbf{f}^{-e(S)/2}+\dots\right),$$

where "…" means that the exponent of the other terms is smaller in modulus (and they give smaller contributions to the asymptotic enumeration). Observe that $\mathbf{g}(z,u)$ is analytic at $(z,u) = (1/4,1)$, and satisfies that $\mathbf{g}_S(z,1) = g_S(z)$. This presentation for $\mathrm{P}_\Sigma(z,u)$ is the correct one to deal with the operator Θ^l: observe that the greatest contribution (using the Transfer Theorem) comes from cubic maps, which are the ones with maximize the number of edges (i.e., the value $e(S) = 3\gamma(\Sigma)+2\beta(\Sigma)-6$). For conciseness on the formulas, until the end of this section we write $e = 3\gamma(\Sigma)+2\beta(\Sigma)-6$.

We need to study the derivative $\frac{\partial^l}{\partial u^l}\left(\mathbf{g}_S(z,u)\mathbf{f}^{-e/2}\right)$, which is the main contribution of each cubic map. When we apply this derivative over $\mathbf{g}_S(z,u)\mathbf{f}^{-e/2}$, the greatest contribution comes from $\mathbf{g}_S(z,u)u^l\frac{\partial}{\partial u^l}\mathbf{f}^{-e/2}$, because this term maximizes the exponent (in modulus) of the singular term. In this case, the singular term with greatest exponent corresponds to

$$u^l\mathbf{g}_S(z,u)\frac{(-1)^l\Gamma(e/2+l)}{\Gamma(e/2)}\frac{(\mathbf{f}_u(z,u))^l}{\mathbf{f}(z,u)^{e/2+l}},$$

where $\mathbf{f}_u(z,u)$ is the derivative of $\mathbf{f}$ with respect to u. Writing $u=1$, the previous expression is simplified into

$$g_S(z)\frac{\Gamma(e/2+l)}{\Gamma(e/2)}\frac{(2z)^l}{(1-4z)^{e/2+l}}.$$

To estimate the value of the k-th coefficient of the previous GF, we apply the Transfer Theorem for singularity analysis (Equation (8.1)), obtaining

$$g_S(1/4)\frac{1}{\Gamma(e/2)}2^{-l}\cdot k^{e/2+l-1}\cdot 4^k(1+O(k^{-1/2})).$$

To conclude, recall that this above term is the contribution of a single cubic map. Summing over all cubic maps, we obtain the following proposition:

Proposition 8.7 *Let $p^\Sigma_{k,r}$ be the number of irreducible 2-zone decompositions of Σ. For a fixed positive integer l, the following asymptotic approximation holds:*

$$\sum_{r=1}^{k} r^l p^\Sigma_{k,r} =_{k\to\infty} \frac{C(\Sigma)\cdot 2^{-l}}{\Gamma(3\gamma(\Sigma)/2+\beta(\Sigma)-3)}\cdot k^{3\gamma(\Sigma)/2+\beta(\Sigma)-4+l}\cdot 4^k(1+O(k^{-1/2})), \tag{8.14}$$

where an upper bound for $C(\Sigma)$ is stated in Proposition 8.4.

8.13 Bell Structures: from Partitions to Packings

For a fixed number of k vertices, fix an arbitrary subset of $i\le k$ vertices, and consider the set of non-crossing partitions over Σ using this set of i vertices. This value is precisely p^Σ_i. This set of i vertices can be chosen in $\binom{k}{i}$ ways. Consequently, we want to estimate the sum $\sum_{i=0}^k\binom{k}{i}p^\Sigma_i$. Observe that this construction is quite close to Bell numbers, which count the number of ways a set of k elements can be partitioned into nonempty subsets. The main result of this section uses the following combinatorial trick:

Lemma 8.12 *Let $\boldsymbol{A}(z)=\sum_{n>0}a_nz^n$. Then the sum $\sum_{i=0}^n\binom{n}{i}a_i$ is $[z^n]\frac{1}{1-z}\boldsymbol{A}(\frac{z}{1-z})$.*

Proof: It is a consequence of the Taylor development of $\frac{1}{1-z}\mathbf{A}(\frac{z}{1-z})$ and the relation $\frac{z^n}{(1-z)^{n+1}}=\sum_{i=0}^\infty\binom{n+i}{i}z^{n+i}$, which can be proved by induction. □

Consequently, $\mathrm{P}_\Sigma(z,1)$ is modified via Lemma 8.12 to obtain the GF for the numbers $\sum_{i=0}^k\binom{k}{i}p^\Sigma_i$. The singularity of $\mathrm{P}_\Sigma(z,1)$ is located at $z=1/4$, and therefore the singularity of $\frac{1}{1-z}\mathrm{P}_\Sigma(z/(1-z),1)$ is located at $z=1/5$. Its singular behavior (i.e., the singular exponent) is the same as the one for $\mathrm{P}_\Sigma(z,1)$. The modification is made only on the position of the singularity, and not on its nature.

Summarizing, the estimate of $\sum_{i=0}^k\binom{k}{i}p^\Sigma_i$ for k big enough has exponential term equal to 5^k, and subexponential term equal to the one of p^Σ_k. In other words, we have proved the following proposition:

Proposition 8.8 *The following estimate holds:*

$$\frac{\sum_{i=0}^k\binom{k}{i}p^\Sigma_i}{p^\Sigma_k} =_{k\to\to\infty}\left(\frac{5}{4}\right)^{k+1}\left(1+O(k^{-1/2})\right).$$

8.14 Conclusions

Our results can be summarized as follows.

Theorem 8.4 *Given a problem P belonging to Category (C) in a graph G embedded in a surface of Euler genus γ, with $\mathbf{bw}(G) \leq k$, the size of the tables of a dynamic programming algorithm to solve P on a surface cut decomposition of G is bounded above by $2^{O(k)} \cdot k^{O(\gamma)} \cdot \gamma^{O(\gamma)}$.*

As we mentioned, the problems tackled in [97] are those in Category (B), which are included in Category (C). As a result of this, we reproduce all the results of [97]. Moreover, as our approach does not use planarization, our analysis provides algorithms where the dependence on the Euler genus γ is better than the one in [97]. In particular, the running time of the algorithms in [97] is $2^{O(\gamma \cdot \mathbf{bw} + \gamma^2 \cdot \log(\mathbf{bw}))} \cdot n$, while in our case the running time is $2^{O(\mathbf{bw} + \gamma \cdot \log(\mathbf{bw}) + \gamma \cdot \log \gamma)} \cdot n$.

Dynamic programming is important for the design of subexponential exact or parameterized algorithms. Using the fact that bounded-genus graphs have branchwidth at most $O(\sqrt{\gamma \cdot n})$ [122], we derive the existence of exact algorithms in $O^*(2^{O(\sqrt{\gamma n} + \gamma \cdot \log(\gamma \cdot n))})$ steps for all problems in Category (C). Moreover, using bidimensionality theory (see [92, 94]), on can derive $2^{O(\gamma \cdot \sqrt{k} + \gamma \cdot \log(\gamma \cdot k))} \cdot n^{O(1)}$ step parameterized algorithms for all bidimensional problems in Category (C).

A natural extension of our results is to consider more general classes of graphs than bounded-genus graphs. This has been done in [99] for problems in Category (B), where the tables of the algorithms encode pairings of the middle set. To extend these results for problems in Category (C) (where tables encode subsets of the middle set), using the planarization approach of [99], appears to be a quite complicated task. We believe that our surface-oriented approach could be more successful in this direction.

Notice that Categories (A), (B), and (C) can be seen as the first levels of a more general hierarchy of dynamic programming algorithms designed for gradually more complicated combinatorial problems. For instance, higher level classes of algorithms can be defined for tables encoding connected pairings (or even connected packings) of subsets of the middle set. In a sense, what we prove in this chapter is the collapse of the time bounds in Category (C) to those in Category (A) when inputs are topologically restricted. It seems to be an interesting task to define such a hierarchy and to check whether this collapse extends to its higher levels.

Part IV

Conclusions and Further Research

The conclusions and avenues for further research corresponding to each chapter of this thesis have been given in page 46 (Chapter 1), page 65 (Chapter 2), page 91 (Chapter 3), page 113 (Chapter 4), page 144 (Chapter 5), page 162 (Chapter 6), page 173 (Chapter 7), and page 208 (Chapter 8). From a more global point of view, here we conclude the thesis and briefly propose possible lines for further research in Sections IV.1 and IV.2, respectively.

IV.1 Final conclusions

This thesis consisted of two main parts: traffic grooming and degree-constrained subgraph problems. Originally motivated by an optimization problem in optical networks (Chapters 1-4), we got interested in progressively more general problems. Namely, in Chapters 5-7 we focused on degree-constrained subgraph problems, and in Chapter 8 we provided a framework to deal with problems whose solutions can be codified by subsets of vertices. Let us now give some more details about the problems we considered.

Traffic grooming. In Chapter 1 we studied the traffic grooming problem on rings and paths with a general traffic pattern. We proved hardness results and provided approximation algorithms. In Chapter 2 we modeled the traffic grooming in unidirectional rings in the case when the request graph has bounded maximum degree and the objective is to design a network being able to support any request graph satisfying the degree constraints. We were able to settle the (asymptotically) optimal number of ADMs at each node for almost all values of the grooming factor and the maximum degree. In Chapters 3 and 4 we focused on the ring topology with an all-to-all traffic pattern. Namely, in Chapter 3 we studied the bidirectional ring, providing optimal solutions for infinite families of values of the size of the network and the grooming factor. In Chapter 4 we dealt with the unidirectional ring and two grooming factors C and C' that alternate dynamically. Using tools from combinatorial designs, we found the optimal switching cost for $C = 4$ and $C' \in \{1, 2, 3\}$, as well as the optimal switching cost under the constraint of using the minimum number of wavelengths.

In view of our results, we observe that there is a trade-off between the *generality* of the considered problem (like the request graph, the topology, or the grooming factor) and the *quality* of the solutions that one can expect to obtain in a reasonable computation time. Namely, the more general the setting is, the further we are from an optimal solution. This phenomenon is not surprising, as traffic grooming is an NP-hard problem and, unless P = NP, optimal solutions of an NP-hard problem cannot be found in polynomial time.

Degree-constrained subgraph problems. In the second part of this thesis we applied a variety of approaches to study degree-constrained subgraph problems. In Chapter 5 we provided hardness results and approximation algorithms for several such problems, using for instance the error amplification technique and randomized algorithms. We studied in Chapter 6 the parameterized complexity of these problems in order the better understand their apparent hardness. We proved $W[1]$-hardness results using parameterized reductions

and provided FPT algorithms using refined dynamic programming and structural results from graph minors theory. In Chapters 7 and 8 we focused on the case when the input is topologically restricted. Namely, in Chapter 7 we obtained subexponential parameterized and exact algorithms for a family of degree-constrained subgraph problems on planar graphs, using bidimensionality theory combined with novel dynamic programming techniques. Finally, we provided in Chapter 8 a framework for the design of algorithms with single-exponential dependence on branchwidth for a broad class of problems on graphs embedded in surfaces. This framework introduces a new type of branch decomposition, called *surface cut decomposition*, and uses tools from topological graph theory together with the symbolic method and singularity analysis from analytic combinatorics.

IV.2 Further Research

Concerning traffic grooming, we have mentioned along the thesis that the unidirectional ring with all-to-all traffic in an important special case. So far, the formulas of the minimum number of ADMs as a function of the network size have been found for values of the grooming factor up to seven (see page 30). Finding the optimal cost for each value of the grooming factor involves complicated tailor-made constructions. Although finding these formulas may yield new combinatorial designs and interesting insights, it does not make sense to aim at solving all the (infinite) cases of the grooming factor one by one. It would be more relevant to conceive a machinery (probably, with the help of a computer) able to generate the cost formulas in reasonable time for each fixed value of the grooming factor.

In this thesis we moved towards more and more general problems. How far can one go in this direction? The answer probably lies on the so-called *algorithmic meta-theorems*. One of the most notorious such theorems, the fundamental theorem of Courcelle [84], states that graph properties definable in monadic second-order logic can be decided in linear time on graphs of bounded treewidth (or equivalently, bounded branchwidth, see page 16). This was the first in a series of algorithmic meta-theorems. More recent examples of such meta-theorems state that all first-order definable properties of planar graphs can be decided in linear time [126] and that all first-order definable optimization problems on classes of graphs with excluded minors can be approximated in polynomial time to any given approximation ratio [88]. The term "meta-theorem" refers to the fact that these results do not describe algorithms for specific problems, but for whole families of problems, whose definition typically has a logical and a structural (usually graph-theoretical) component. For example, Courcelle's theorem [84] is about monadic second-order logic on graphs of bounded treewidth. The general form of algorithmic meta-theorems is: "All problems definable in a *certain logic* on a certain class of *structures* can be solved *efficiently*". We refer the reader to the excellent survey of Grohe [138].

Many of the meta-theorems are tightly linked with graph minor theory. Recently, results from graph minor theory have been combined with algorithmic techniques that had originally been developed for planar graphs to obtain polynomial time approximation schemes and FPT algorithms for many standard optimization problems on families of graphs with excluded minors. The fascinating topic of algorithmic meta-theorems is receiving increasing attention during the last years and seems to be a promising research field.

Appendix A

Permutation Routing and (ℓ, k)-routing on Plane Grids

A.1 Permutation Routing on Triangular Grids

A.2 (ℓ, k)-routing on Plane Grids

Discrete Mathematics and Theoretical Computer Science DMTCS vol. **10:3**, 2008, 49–62

An Optimal Permutation Routing Algorithm on Full-Duplex Hexagonal Networks†

Ignasi Sau[1‡] and Janez Žerovnik[2§]

[1] *Mascotte joint Project- INRIA/CNRS-I3S/UNSA- 2004, route des Lucioles - Sophia-Antipolis, France; and Graph Theory and Combinatorics group at Applied Mathematics IV Department of UPC - Barcelona, Spain.*
email: ignasi.sau@gmail.com
[2] *University of Maribor, FME, Smetanova 17, SI-2000 Maribor, Slovenia; and IMFM, Jadranska 19, SI-1000 Ljubljana, Slovenia.*
email: janez.zerovnik@imfm.uni-lj.si

received 13 March 2008, *accepted 10 July 2008.*

In the permutation routing problem, each processor is the origin of at most one packet and the destination of no more than one packet. The goal is to minimize the number of time steps required to route all packets to their respective destinations, under the constraint that each link can be crossed simultaneously by no more than one packet. We study this problem in a hexagonal network, i.e. a finite subgraph of a triangular grid, which is a widely used network in practical applications.
We present an optimal distributed permutation routing algorithm on full-duplex hexagonal networks, using the addressing scheme described by F.G. Nocetti, I. Stojmenović and J. Zhang (*IEEE TPDS 13(9): 962-971, 2002*). Furthermore, we prove that this algorithm is oblivious and translation invariant.

Keywords: hexagonal networks, permutation routing, shortest path, distributed algorithm, communication networks, oblivious algorithm.

1 Introduction

The packet-routing problem on any interconnection network is essentially important. This problem involves how to transfer the right data to the right place within a reasonable amount of time. To measure the routing capability of an interconnection network, the partial permutation routing (PPR) problem is usually used as the metric. In the PPR problem, each processor is the origin of at most one packet and the destination of no more than one packet. This problem has been studied in a wide diversity of scenarios,

†A short conference version of this article (with incomplete proofs) was presented in Sau and Žerovnik (2007).

‡Partially supported by European project IST FET AEOLUS, PACA region of France, Ministerio de Educación y Ciencia of Spain, European Regional Development Fund under project TEC2005-03575, Catalan Research Council under project 2005SGR00256, and COST action 293 GRAAL, and CRC CORSO with France Telecom.

§Supported in part by the Slovenian research agency ARRS, grants L2-7207-0101 and P1-0285-0101.

such as Mobile Ad Hoc Networks Karimou and Myoupo (2005), Cube-Connected Cycle (CCC) Networks Jan and Lin (2005), Wireless and Radio Networks Datta (2005), All-Optical Networks Liang and Shen (2002), and Reconfigurable Meshes Cogolludo and Rajasekaran (2001).

Recently an optimal algorithm for permutation routing has been found on full-duplex 2-circulant graphs Hwang et al. (1997). Routing algorithms for 2-circulants with half-duplex links are studied in Dobravec et al. (2003). In this paper we give an optimal algorithm for permutation routing on full-duplex hexagonal networks.

Two-dimensional meshes are among the most studied topologies for computer networks. Tessellation of the plane with hexagons may be considered as the most natural because cells have optimal diameter to area ratio. If centers of neighboring cells are connected, one obtains a triangular grid. Hexagonal networks are finite subgraphs of the triangular grid, and in this article we study convex subgraphs (that is, that contain all shortest paths between any pair of nodes) of the triangular grid. The triangular grid can also be obtained from the basic 4-mesh by adding NE to SW edges, which is called a 6-mesh in Trobec (2000).

Another suitable application for this problem can be easily found on a radiocommunication wireless environment Nocetti et al. (2002). Let the base stations be placed at the centers of a hexagonal tessellation of the plane. The interconnection network among base stations constitutes a hexagonal network, as shown in Figure 1.

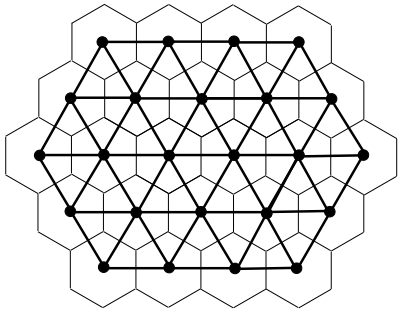

Figure 1: Hexagonal network (△) and hexagonal tessellation (⬡).

Coordinates must be defined to associate to each mobile user the base station which is the center of the hexagon where this user is. The problem of exchanging messages among mobile users corresponds to a routing problem, since there are pairs of users willing to communicate. Each user can only establish one call at a given moment. If one assumes that in each hexagonal cell there is at most one user that sends a message and at most one user that receives a message, the problem can be modeled as partial permutation routing. The algorithm that we propose can be used to route the messages between pairs of users that are in different hexagons. If there are more messages to be routed, i.e. there are more than one message originating from the same cell or a cell is the destination of more than one message, our algorithm still can be used, but the optimality may not be achieved. If the two communicating users belong to the same hexagon, some local delivery mechanism can be carried out.

This paper is structured as follows. In Section 2 we define preliminary concepts that will be used later. The description and analysis of our algorithm are provided in Section 3. Finally, Section 4 concludes this work.

2 Preliminaries

In this section we describe the network topology under study, we formally define the routing problem and we recall previous results on this field.

2.1 Network topology

Nodes in a hexagonal network are placed at the vertices of a regular triangular tessellation, so that each node has up to six neighbors. These networks have been studied in a variety of contexts, specially in wireless and interconnection networks Decayeux (2006); Decayeux and Seme (2005), but they have been also applied, for instance, in chemistry to model benzenoid hydrocarbons Tošić et al. (1995).

We focus on the hexagonal network with full-duplex links, that is, an edge of the network can be crossed simultaneously by two messages, one in each direction. Equivalently, each edge between two nodes u and v is made of two independent arcs $\{uv\}$ and $\{vu\}$, as illustrated in Figure 2a.

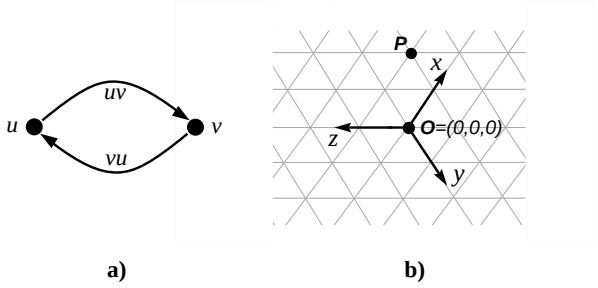

Figure 2: a) Each edge consists of two independent links. **b)** Axis used in a hexagonal network.

Remark 2.1 *If the network is half-duplex, it is easy to construct a 2-approximation algorithm from an optimal algorithm for the full-duplex case by introducing* odd-even *steps, as explained for example in Dobravec et al. (2003).*

2.2 Routing problem

We deal with a partial permutation routing problem. In an infinite triangular grid, we are given a subset V of nodes, and a permutation π that acts on this subset $\pi : V \to V$. Each node $u \in V$ wants to send a message to the node $\pi(u)$. Thus, we have $|V|$ pairs of communicating nodes and $|V|$ messages to be delivered simultaneously. All the edges and nodes of the *host* graph (the hexagonal graph) can be traversed by the packets.

Remark 2.2 *The special case where only one node has a packet to send is known as* 2-terminal routing *(*minimum *if a shortest path is used). Optimal algorithms for the 2-terminal routing problem have been found, for instance, in 2-jump circulant graphs Robič and Žerovnik (2000) and hexagonal networks Nocetti et al. (2002).*

Under the store-and-forward Δ-port model Fraigniaud and Lazard (1994), at each step a packet can either stay or move to an adjacent node by crossing a link, but no link can be crossed by two packets at the same step (recall that in the full-duplex case, there are two links between two adjacent nodes). A node can send or receive packets through all its incident edges at the same step. Cohabitation of multiple packets

at the same node is allowed. Thus, a queue is required for each outgoing edge at each node. Since the outdegree in a hexagonal network is six, the same number of queues are required at each node. The goal is to minimize the number of time steps required to route all packets to their respective destinations.

The algorithm described in Section 3 is implemented independently at each node, without assuming any global knowledge about the network. I.e., it is a fully distributed algorithm.

Remark 2.3 *One could also have defined the permutation routing with a permutation π that acts on all the nodes of an infinite hexagonal network, defining the (infinite set of) non-communicating nodes as fixed points of π.*

2.3 Addressing model and previous results

In Nocetti et al. (2002) the authors solve the problem of routing a *single* message following a shortest path through a hexagonal communication network. The first idea that is of our interest (in fact, this was first introduced in Stojmenović (1997)) is the representation of any address on a generating system consisting of three unitary vectors $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ on the directions of three axis x, y, z with a 120 degree angle between each pair. These three axes intersect on an arbitrary (but fixed) node O labeled with the address $\boldsymbol{O} = (0, 0, 0)$, as it is depicted in Figure 2b. An example will be given later. Clearly any path on the triangular grid is a sequence of moves in directions **i**, **j**, **k**, or in the opposite directions. From commutativity of vector addition it follows that any path can be expressed as a combination $P_1\mathbf{i} + P_2\mathbf{j} + P_3\mathbf{k}$. Hence each node P of the hexagonal network can be labeled with an address $\boldsymbol{P} = (P_1, P_2, P_3)$ based on paths from the origin O. Using that $\boldsymbol{i} + \boldsymbol{j} + \boldsymbol{k} = 0$, the key observation Nocetti et al. (2002) is that, if (a, b, c) and (a', b', c') are the relative addresses of two packets, then $(a, b, c) = (a', b', c')$ if and only if there exists $d \in \mathbb{Z}$ such that $a' = a + d$, $b' = b + d$, and $c' = c + d$. For instance, consider the node P of Figure 2b. We can express its address $\boldsymbol{P}$ in different equivalent ways, for instance: $\boldsymbol{P} = (3, 1, 2) = (2, 0, 1) = (1, -1, 0) = (0, -2, -1)$. Among many possible paths and corresponding addresses we will later use the unique address called the address in the shortest path form. As we will see in Section 3, at the beginning of the routing algorithm each node S knows the address of the destination node D of the message placed initially at S, and computes the relative address $\overrightarrow{SD} = \boldsymbol{D} - \boldsymbol{S}$ of the message. Note that this relative address does not depend on the choice of the origin node O. As discussed later, this relative address is the only information that is added to the heading of the message to be transmitted, constituting in this way the packet to be sent through the network. The relative address $\overrightarrow{OP} = \boldsymbol{P} - \boldsymbol{O}$ can be expressed in different ways. Note that these addresses are related to paths of different lengths.

Definition 2.1 *A relative address $\overrightarrow{SD} = (a, b, c)$ is of the* shortest path form *if there is a path from node S to node D, consisting of a units of vector* **i**, *b units of vector* **j** *and c units of vector* **k**, *and this path has the shortest length among all paths going from S to D.*

In what follows, (a, b, c) denotes a relative address. We will later work only with the addresses in the shortest path form. The next result simplifies extraordinarily the routing in hexagonal networks, as we will see in Section 3.

Theorem 2.1 (Nocetti et al. (2002)) *An address (a, b, c) is of the shortest path form if and only if at least one component is zero, and any two components do not have the same sign.*

In the example above, one can check that the address $(1, -1, 0)$ has minimum length, and it is the only one that satisfies the conditions of Theorem 2.1.

Corollary 2.1 (Nocetti et al. (2002)) *Any address has a unique shortest path form.*

Thus, each relative address $\overrightarrow{SD}$ written in the shortest path form has at most two non-zero components, and they have different sign. In fact, it is easy to find the shortest path form (and thus, the length) using the next result.

Theorem 2.2 (Nocetti et al. (2002)) *If $\overrightarrow{SD} = a\mathbf{i}+b\mathbf{j}+c\mathbf{k}$, then the length of the path given by the relative address $\overrightarrow{SD}$ satisfies*

$$|\overrightarrow{SD}| = \min(|a-c|+|b-c|, |a-b|+|b-c|, |a-b|+|a-c|).$$

3 An optimal routing algorithm

In Section 3.1 we describe our algorithm for permutation routing in full-duplex hexagonal networks, and in Section 3.2 we prove that this algorithm is optimal. Informally, the idea of this algorithm is to define a routing for each type of shortest path and a suitable queue policy, in such a way that the number of steps that a packet has to wait plus the length of its shortest path (from its origin to its destination) do not exceed the bound given by the maximum length over the paths of all packets to be sent.

3.1 Formal description

We introduce some definitions in order to simplify further proofs.

Definition 3.1 *Given a packet p and its relative address (a,b,c) in the shortest path form, we denote by ℓ_p the length of this shortest path, and by ℓ_{max} the maximum length over all shortest paths:*

$$\ell_p := |a| + |b| + |c|\,, \qquad \ell_{max} := \max_p(\ell_p)$$

Since $|V| < \infty$, this maximum is indeed achieved, possibly by several messages. By the definition of ℓ_{max} and taking into account that any algorithm can move a packet only one position at each step, ℓ_{max} is in fact a lower bound for all algorithms.

Lemma 3.1 (Lower Bound) *The number of steps of any permutation routing algorithm is at least ℓ_{max}.*

Definition 3.2 *Two packets p and p' are* in conflict *or, simply,* meet*, if they are simultaneously (i.e. on the same step of the algorithm) in the same outgoing queue at the same node of the network.*

Intuitively, if two (or more) packets meet, at most one of them moves on the next step of the algorithm.

Definition 3.3 *We denote by w_p^i the number of steps waited by packet p until the end of the i*th *step of the algorithm. We call w_p^i the* delay of *p.*

Given a packet p and its delay w_p^i, w_p^i is an allowed delay *if $\ell_p + w_p^i \leq \ell_{max}$. Similarly, w_p^i is an* additional *(or* forbidden*)* delay *if $\ell_p + w_p^i > \ell_{max}$.*

Finally, a packet p is saturated *at the end of step i if $w_p^i = \ell_{max} - \ell_p$.*

The idea is that if a packet p is saturated it must not wait anymore. Otherwise, the algorithm becomes not optimal. In Figure 3 the packet model that we use for the analysis is represented. In this model, each packet has two *boxes* with capacities ℓ_p and $\ell_{max} - \ell_p$, respectively. At each step of the algorithm, two things can happen. Namely, if packet p has moved during step i, then an item of the box on the left (first

box) is filled. On the other hand, if packet p has waited in some queue during step i, then an item of the box on the right (second box) is filled. If the first box is full, it means that packet p has reached its destination. Conversely, if the second box is full, it means that packet p is *saturated*, and thus it cannot wait anymore if we want to guarantee the optimality of our algorithm (because the number of steps required by p would be strictly greater than ℓ_{max}).

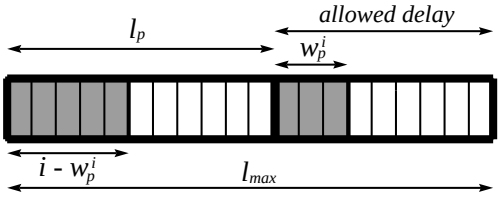

Figure 3: Packet model used on the permutation routing algorithm.

Remark 3.1 *The model displayed in Figure 3 is only used for analysis, but in fact the only information that each packet p has to carry is the relative address until its destination. That is, only a vector with at most two non-zero integers. It is important to clarify that it is not necessary that the global constant ℓ_{max} is available at every node, because some broadcasting protocol would have to be carried out to compute and share it. Although it is possible to broadcast on hexagonal mesh in polynomial time Chen et al. (1990), this would introduce additional time complexity in the preprocessing phase.*

Table 1 describes an optimal message routing algorithm for each node P. Note that the main loop for is infinite. We prove that in the case of permutation routing, the number of steps i at each node is at most ℓ_{max}, but written in this way the algorithm can be applied in a more general routing scenario.

Remark 3.2 *We assume that the network has a global clock, and that all nodes are synchronized. Packets can be sent only on discrete clock events, while the other tasks (**preprocessing**, **update_packet**, **decide_outgoing_edge and order_queue**) are done between two consecutive clock events.*

In next sections we provide a detailed explanation of each of the procedures that appear on this general scheme.

3.1.1 Description of the procedure **preprocessing**

Before packets begin to be sent through the network, at most one packet is placed at each node. Each packet has associated the address of its destination node, namely D. With this information, each source node S can compute the relative address (a, b, c) of the packet that will be sent from S to D. Because of Theorem 2.1, to write this address in the shortest path form it is enough to check which address among $(0, b-a, c-a)$, $(a-b, 0, c-a)$ and $(a-c, b-c, 0)$ has two components of different sign.

Then each node computes the length of each packet's address (a, b, c). As it is in the shortest path form, the path that the packet will have to follow has length $\ell_p = |a| + |b| + |c|$. Finally, for each message a heading containing the components of the relative address is added. Note that this information is enough considering the packet model of Figure 3, as we prove in Lemma 3.4.

Thus, since this task involves just integer addition and comparison, the time complexity of the preprocessing phase is $\mathcal{O}(1)$ assuming fixed integer size to codify all addresses, or $\mathcal{O}(\log(n))$, where n is the maximum size of the integer required to codify the addresses of the nodes.

```
        At each node P of the network:

 1 :  begin
 2 :      preprocessing;
 3 :      for i from 0 do
 4 :          Reception phase:
 5 :              for each packet in node P
 6 :                  update_packet;
 7 :          Transmission phase:
 8 :              for each packet in node P
 9 :                  decide_outgoing_edge;
10 :              for each queue
11 :                  order_queue;
12 :                  send the 1st packet;
13 :      increment i
14 :      end for
15 :  end
```

Table 1: Algorithm $\mathcal{A}$.

3.1.2 Description of the procedure **update_packet**

Updating the address in reception offers more robustness against link failures. Of course, a node knows from which neighbor a packet comes from. Without ambiguity, we assign to each incoming edge a vector as shown in Figure 4a. (Formally, in Table 2 *incoming_edge* is the vector assigned to the edge used by the incoming packet. The same idea applies to *outgoing_edge* in Table 3.) Then to update the relative address (a, b, c) of the packet, one has just to decrease by one the component corresponding to the incoming edge, as described in Table 2. After updating the address, we check if the packet has reached its destination node.

```
1 :  begin
2 :      (a, b, c) ← (a, b, c) − incoming_edge;
3 :      if (a, b, c) == (0, 0, 0)
4 :          then this is the destination node;
5 :  end
```

Table 2: Description of the procedure **update_packet**.

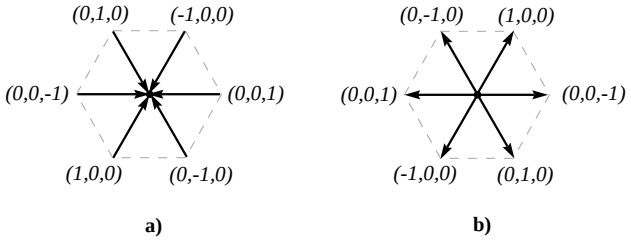

Figure 4: a) Possible incoming edges to (0,0,0). **b)** Possible outgoing edges from (0,0,0).

3.1.3 *Description of the procedure* **decide_outgoing_edge**

This is the phase where the main core of the routing is carried out. The idea of this routing is that one can assure that when packets meet, no additional delay is introduced, as we prove in Proposition 3.1. Given the address of the shortest path of a packet with two non-zero components, one can choose one of both directions to begin with, as we see with an example in Figure 5. This procedure makes the choice between both alternatives in each case.

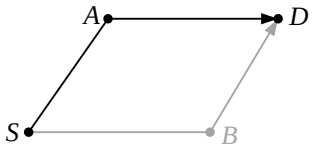

Figure 5: Given the relative address of the packet, there are two possible choices for assigning priorities to the components of the shortest path.

Because of Theorem 2.1, we can partition the $|V|$ relative addresses in the shortest path form into 12 disjoint classes, according to the sign of their non-zero components. These classes are, namely:

$$(+,0,0),(-,0,0),(0,+,0),(0,-,0),(0,0,+),(0,0,-),$$
$$(+,-,0),(-,+,0),(+,0,-),(-,0,+),(0,+,-),$$
$$\text{and } (0,-,+).$$

Note that the address of a packet may vary at each step, as we have seen in the **update_packet** procedure. It is also interesting to observe that the same triple can appear many times as more pairs can have the same relative addresses. However, at a given time step, all relative addresses of packets that appear at the same node are different.

Recall that in a hexagonal network each node has six outgoing edges, and thus six queues. Without ambiguity, we label the six outgoing edges of a node as shown in Figure 4b. For the updated packet address (a,b,c), Table 3 describes this routing procedure.

For instance, if the packet address is of the type $(-,0,+)$ then, according to Table 3, this packet goes first in the direction $-x$, and after in the direction $+z$. (See Example 3.1.) We symbolize this rule by

```
1 : begin
2 :    if packet_address has only 1 non-zero component
3 :       outgoing_edge = the edge corresponding
                          to the non-zero component;
4 :    else
5 :       outgoing_edge = the edge corresponding
                          to the negative component;
6 : end
```

Table 3: Description of the procedure **decide_outgoing_edge**.

the arrow ↙. Another example: the routing of the address $(+,-,0)$ is represented by ↖. That is, we always give priority to the negative component. In Figure 6 these routing rules are summed up.

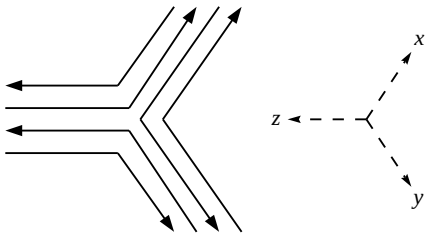

Figure 6: Routing of the packets according to Algorithm $\mathcal{A}$.

Definition 3.4 *Given a packet p and its relative address (a,b,c) with two non-zero components, we call the* first *(resp.* second*)* direction of p *to the first (resp. second) direction of the movement of the message according to the rules of Algorithm $\mathcal{A}$. If $\overrightarrow{SD}$ has only one non-zero component, we say that it is* the direction of p.

Example 3.1 *We illustrate the operation of procedures **decide_outgoing_edge** and **update_packet**. Assume a packet with relative address $(-1,0,4)$ is given to the procedure **decide_outgoing_edge**. As it has two nonzero components, it is added to the queue corresponding to the negative component, i.e. $(-1,0,0)$. After possibly waiting some communication rounds in the queue, the packet will be transmitted. After arrival to the next node, first its relative address will be altered by the procedure **update_packet** to $(0,0,4)$, which is $(-1,0,4)-(-1,0,0)$. Now the relative address has only one nonzero component, and therefore the procedure **decide_outgoing_edge** will send it to the queue of this direction, i.e. $(0,0,1)$.*

3.1.4 Description of the procedure **order_queue**

At each of the six queues of each node, the same priority policy applies: order the outgoing packets according to *decreasing number of remaining steps until the destination*. That is, the packet that has more remaining steps has priority 1, the next one has priority 2, and so on. Let us show in Lemma 3.2 and Lemma 3.3 that there cannot be equality in the number of remaining steps, hence this policy is correct.

Lemma 3.2 *Packets can only wait, possibly, during their last direction.*

Proof: Suppose that the claim is false, and consider the first (according to the running time of the algorithm) two packets having a relative address with two non-zero components that meet when their first direction is not yet finished. According to the routing rules of **decide_outgoing_edge** procedure (see Table 3 or Figure 6), the direction where packets met must be the same: either $-x$, $-y$, or $-z$. Since these packets are the first packets that meet, both must have the same origin node, a contradiction. □

Note that one can generalize this result by saying that a packet can wait only during its *last* direction, because packets with only one direction can be in conflict obviously only on this direction. We are now ready to prove that the queue policy that we have defined is correct:

Lemma 3.3 *Two packets in a given queue cannot have the same number of remaining steps.*

Proof: Let p and p' be two packets in the same queue. Because of Lemma 3.2, both p and p' must be in their last direction. If they had the same number of remaining steps, they would have the same destination node, a contradiction. □

3.2 Proof of optimality

The next observation is useful in the proof of Proposition 3.1, which is the main result that allows us to prove the optimality of Algorithm $\mathcal{A}$.

Lemma 3.4 *The following two queue policies are equivalent:*

1. *Order the outgoing packets according to decreasing remaining steps.*
2. *Order the outgoing packets according to increasing remaining allowed delay.*

Proof: For any packet p (see the model of Figure 3) the number of remaining steps is $\ell_p - (i - w_p^i) = \ell_p - i + w_p^i$, and the remaining allowed delay is $\ell_{max} - \ell_p - w_p^i$. Thus, without taking the sign into account, both magnitudes differ only on i and ℓ_{max}, which are constants for all packets at a given step i of the algorithm. □

Proposition 3.1 *Algorithm $\mathcal{A}$ introduces no additional delay to any packet p.*

Proof: We prove by induction on the number of steps that no additional delay is introduced at any queue of the network after step i.

First of all, after the first step ($i = 1$) the only packets that could have additional delay are those with length ℓ_{max}. But, since all those nodes must be placed into different nodes at the beginning of the

algorithm, no two nodes can have been in the same queue, and thus all of them must have had maximum priority according to the queue policy, hence none has waited.

Now we suppose that no additional delay has been introduced after step $i-1$ ($i \geq 2$) and let us show that no additional delay is introduced after step i. It is enough to see that there can be at most one saturated packet at each queue. Indeed, if there is only one saturated packet, this packet has maximum priority because of Lemma 3.4, and thus this packet does not wait. Now, suppose that there are two saturated packets p and p' in the same queue. By definition of saturated packet, $\ell_p + w_p^i = \ell_{p'} + w_{p'}^i = \ell_{max}$. The number of remaining steps of p is $\ell_p - (i - w_p^i) = \ell_{max} - i$, which equals the number of remaining steps of p': $\ell_{p'} - (i - w_{p'}^i) = \ell_{max} - i$. Thus, both packets have the same destination node, contradicting the assumption of permutation routing. □

Theorem 3.1 *Algorithm $\mathcal{A}$ is an optimal permutation routing algorithm for full-duplex hexagonal networks.*

Proof: Because of Proposition 3.1, all packets reach their destination nodes in a number of steps not greater than ℓ_{max}, and thus the lower bound of Lemma 3.1 is attained. □

Besides minimizing the number of steps, a routing algorithm must also be easy to implement; namely, the routing at each step should be determined efficiently. This class of algorithms is called *oblivious* Hwang et al. (1997, 2002).

Definition 3.5 *A routing algorithm is called* oblivious *if the routing of a packet only depends on its origin and destination nodes, although the waiting time at any intermediate node may depend on other paths. An oblivious algorithm is* translation invariant *if the path between any two nodes u and v depends only on the relative address from u to v (i.e. the path does not depend on the absolute addresses of u and v).*

Thus, a translation invariant oblivious algorithm is completely determined by paths from any node to all other nodes.

Corollary 3.1 *Algorithm $\mathcal{A}$ is oblivious, translation invariant and minimum permutation routing.*

Proof: Optimality, i.e. minimum permutation routing, has been proved in Theorem 3.1. The obliviousness is straightforward since our algorithm only uses the origin and destination nodes for routing each packet. Finally, it is clear that to route a packet only the relative address $\overrightarrow{SD}$ between the source and destination node is necessary, and thus the invariance is also proved. □

Concerning the complexity of the algorithm, at each step each node has to carry out just constant time computations: integer addition and comparison. As stated in Theorem 3.1, the number of time steps is ℓ_{max}. Hence the time complexity of Algorithm $\mathcal{A}$ is $\mathcal{O}(\ell_{max})$.

3.3 About the queue policy

The queue policy plays an important role on the description of our routing algorithm. Another possibility could have been to order the packets according to their *total length* ℓ_p, and then, in case of equality, according the their remaining steps. Note that there is no ambiguity since all destination nodes are different. Although both policies seem to be similar, the total length policy combined by the routing rules shown in Figure 6 yield a non-optimal algorithm, while the remaining steps policy is optimal, as it has been proved

in Theorem 3.1. Indeed, let us show with a counterexample that the total length policy is not optimal. In Figure 7 the origin nodes for each packet are labeled with small letters, while their corresponding destination nodes are labeled with capital letters. We have that $\ell_{A-a} = 6$, $\ell_{B-b} = 5$, $\ell_{C-c} = 5$ and $\ell_{P-p} = 4$. One can check that the total number of steps required by the packet that starts at p to reach P is 7 (because it waits 3 steps) and thus this algorithm is not optimal, since $7 > 6 = \ell_{max}$.

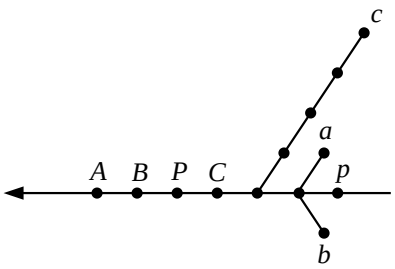

Figure 7: Counterexample to the *total length* queue policy.

4 Conclusions

In this article we have presented a distributed optimal permutation routing algorithm for full-duplex hexagonal networks. Furthermore, we have proved that this algorithm is oblivious and translation invariant. In this way, we have given an answer to an open question proposed in Nocetti et al. (2002). The time complexity of our algorithm is $\mathcal{O}(\ell_{max})$, where ℓ_{max} is the maximum length over the shortest paths of all the packets to be sent.

The next natural step is to consider half-duplex hexagonal networks and honeycomb (half and full-duplex) networks. Another avenue of further research could address generalization of the optimal permutation routing algorithms to more general networks. For example, generalization from 2-circulant graphs to 3-circulant graphs, k-circulants, or, more general, Cayley graphs. Hexagonal networks have a generalization in the three-dimensional hexagonal network García et al. (2003), where the *basic* pieces that *tessellate* the space are tetrahedrons. One could try to find a basis such that all vectors add up to $\vec{0}$, as it happens on the plane. But, unfortunately, one hits on the well known topological result about the non-orientability of the surface of a sphere (a tetrahedron is topologically equivalent to a sphere). Thus, new strategies should be devised for the 3D scenario.

Acknowledgements

We thank the anonymous referees for careful reading and helpful remarks. Most of this work was done while one of us (I.S.) was visiting Institute of Mathematics, Physics and Mechanics in Ljubljana, supported by STSM grant sponsored by European Action COST 293 (GRAAL).

References

M. Chen, K. Shin, and D. Kandlur. Addressing, routing and broadcasting in hexagonal mesh multiprocessors. *IEEE Transactions on Computers*, 1(C-39):10–18, 1990.

J. Cogolludo and S. Rajasekaran. Permutation routing on reconfigurable meshes. *Algorithmica*, 31:44–57, 2001.

A. Datta. A fault-tolerant protocol for energy-efficient permutation routing in wireless networks. *IEEE Transactions on Computers*, 54(11):1409–1421, 2005.

C. Decayeux. *Réseaux hexagonaux: Modélisation Géomètrique et Application à la Téléphonie Mobile*. PhD thesis, Université de Picardie Jules Verne, October 2006.

C. Decayeux and D. Seme. 3d hexagonal network: Modeling, topological properties, addressing scheme, and optimal routing algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 16(9):875–884, 2005.

T. Dobravec, J. Žerovnik, and B. Robič. Permutation routing in double-loop networks: design and empirical evaluation. *Journal of Systems Architecture*, 48:387–402, 2003.

P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. *Discrete Applied Mathematics*, 53:79–134, 1994.

F. García, J. Solano, I. Stojmenović, and M. Stojmenović. Higher dimensional hexagonal networks. *Journal of Parallel and Distributed Computing*, 63(1164-1172), 2003.

F. Hwang, T. Lin, and R. Jan. A permutation routing algorithm for double loop network. *Parallel Processing Letters*, 7(3):259–265, 1997.

F. Hwang, Y. Yao, and B. Dasgupta. Some permutation routing algorithms for low-dimensional hypercubes. *Theoretical Computer Science*, 270:111–124, 2002.

G. E. Jan and M.-B. Lin. Concentration, load balancing, partial permutation routing, and superconcentration on cube-connected cycles parallel computers. *Journal of Parallel and Distributed Computing*, 65: 1471–1482, 2005.

D. Karimou and J. F. Myoupo. An application of an initialization protocol to permutation routing in a single-hop mobile ad hoc networks. *The Journal of Supercomputing*, 31:215–226, 2005.

W. Liang and X. Shen. Permutation routing in all-optical product networks. *IEEE Transactions on Circuits and Systems*, 49(4):533–538, 2002.

F. G. Nocetti, I. Stojmenović, and J. Zhang. Addressing and routing in hexagonal networks with applications for tracking mobile users and connection rerouting in cellular networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(9):963–971, 2002.

B. Robič and J. Žerovnik. Minimum 2-terminal routing in 2-jump circulant graphs. *Computers and Artificial Intelligence*, 19(1):37–46, 2000.

I. Sau and J. Žerovnik. Optimal permutation routing on mesh networks. In *International Network Optimization Conference*, Spa, Belgium, April 2007. 6 pages.

I. Stojmenović. Honeycomb networks: Topological properties and communication algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 8(10):1036–1042, 1997.

R. Tošić, D. Masulović, I. Stojmenović, J. Brunvoll, B. Cyvin, and S. Cyvin. Enumeration of polyhex hydrocarbons up to h=17. *Journal of Chemical Information and Computer Sciences*, 35:181–187, 1995.

R. Trobec. Two-dimensional regular d-meshes. *Parallel Computing*, 26:1945–1953, 2000.

(ℓ, k)-Routing on Plane Grids *

Omid Amini [1] Florian Huc [2] Ignasi Sau [2, 3] Janez Žerovnik [4, 5]

Abstract

The packet routing problem plays an essential role in communication networks. It involves how to transfer data from some origins to some destinations within a reasonable amount of time. In the (ℓ, k)-routing problem, each node can send at most ℓ packets and receive at most k packets. Permutation routing is the particular case $\ell = k = 1$. In the r-central routing problem, all nodes at distance at most r from a fixed node v want to send a packet to v.

In this article we study the permutation routing, the r-central routing and the general (ℓ, k)-routing problems on plane grids, that is square grids, triangular grids and hexagonal grids. We use the *store-and-forward* Δ-port model, and we consider both full and half-duplex networks. We first survey the existing results in the literature about packet routing, with special emphasis on (ℓ, k)-routing on plane grids. Our main contributions are the following :

1. Tight permutation routing algorithms on full-duplex hexagonal grids, and half duplex triangular and hexagonal grids.
2. Tight r-central routing algorithms on triangular and hexagonal grids.
3. Tight (k, k)-routing algorithms on square, triangular and hexagonal grids.
4. Good approximation algorithms (in terms of running time) for (ℓ, k)-routing on square, triangular and hexagonal grids, together with new lower bounds on the running time of any algorithm using shortest path routing.

All these algorithms are completely distributed, i.e. can be implemented independently at each node. Finally, we also formulate the (ℓ, k)-routing problem as a Weighted Edge Coloring problem on bipartite graphs.

Keywords : Packet routing, distributed algorithm, (ℓ, k)-routing, plane grids, permutation routing, shortest path, oblivious algorithm.

1 Introduction

In telecommunication networks, it is essential to be able to route communications as quickly as possible. In this context, the *packet routing* problem plays a capital role. In this problem we are given a network and a set of packets to be routed through the nodes and the edges of the network graph. A packet is characterized by an origin and a destination node, and typically an edge can be used by no more than one packet at the same time. The objective is to find an algorithm to compute a schedule to route all packets which minimizes the total delivery time. This problem has been widely studied in the literature under many different assumptions. In 1988, Leighton, Maggs and Rao proved in their seminal article [29, 30] the existence of a schedule for routing any set of packets with edge-simple paths on a

1. Max-Planck-Institut für Informatik, Saarbrücken, Germany
2. Mascotte Project - I3S (CNRS/UNSA) and INRIA - Sophia-Antipolis, France
3. Graph Theory and Combinatorics group, MA4, UPC, Barcelona, Spain
4. University of Ljubljana Faculty of Mathematics and Physics (IMFM), jubljana, Slovenia
5. University of Maribor, Maribor, Slovenia

*This work has been partially supported by European project IST FET AEOLUS, PACA region of France, Ministerio de Educación y Ciencia of Spain, European Regional Development Fund under project TEC2005-03575, Catalan Research Council under project 2005SGR00256, and COST action 293 GRAAL, and has been done in the context of the crc Corso with France Telecom.

general network, in optimal time of $\mathcal{O}(C + D)$ steps. Here C is the congestion (maximum number of paths sharing an edge) and D the dilation (length of a longest path) and it is assumed that the paths are given a priori. The proof of [30] used Lovász Local Lemma and was non constructive. This result was further improved in [28] where the same authors gave an explicit algorithm, using the Beck's constructive version of the Local Lemma.

These algorithms to compute the optimal schedule are centralized. Then in [38] Ostrovsky and Rabani gave a distributed randomized algorithm running in $\mathcal{O}(C+D+\log^{1+\epsilon}(n))$ steps. We give a more detailed overview in Section 1.1.

Although these results are asymptotically tight, they deal with a general network, and in many cases it is possible to design more efficient algorithms by looking at specific packet configurations or network topologies. For instance, is it natural to bound the maximum number of messages that a node can send or receive. We focus on this point in Section 1.2, where we will formally define the problem under study in this paper.

On the other hand, the network under study plays a major role on the quality and the simplicity of the solution. For example, in a radio wireless environment, cellular networks are usually modeled by a *hexagonal grid* where nodes represent base stations. The cells of the hexagonal grids have good diameter to area ratio and still have a simple structure. If centers of neighboring cells are connected, the resulting graph is called a *triangular grid*. Notice that hexagonal grids are subgraphs of the triangular grid. We will talk about such networks in Section 1.3. In this paper we focus on the study of the (ℓ, k)-routing problem in convex subgraphs, i.e. that contain all shortest paths between all pairs of nodes of the square, triangular and hexagonal grid.

1.1 General Results on Packet Routing

In this section we provide a fast overview of the state-of-the-art of the general packet routing problem, in both the off-line and on-line settings in Sections 1.1.2 and 1.1.3 respectively, focusing mostly on the latter. We begin by recalling three classical lower bounds for the packet routing problem.

1.1.1 Classical lower bounds

In the packet routing problem, there are three classical types of lower bounds for the running time of any algorithm :

1. **Distance bound** : the longest distance over the paths of all packets (usually called *dilation*, denoted D) constitutes a lower bound on the number of steps required to route all the packets.
2. **Congestion bound** : the *congestion* of an edge of the network is defined as the number of paths using this edge. Then, the greatest congestion over all the edges of the network (denoted C) is also a lower bound on the number of steps, since at each step an edge can be used by at most one packet.
3. **Bisection bound** : Let $G = (V, E)$ be the graph which models the network, and $F \subseteq E$ a cut-set disconnecting G into two components G_1 and G_2. Let m be the number of packets with origin in G_1 and destination in G_2. Then, the number of routing steps used by any algorithm will be at least $\left\lceil \frac{m}{|F|} \right\rceil$.

1.1.2 Off-line routing

Given a set of packets to be sent through a network, a *path system* is defined as the union of the paths that each packet must follow. For a general network and any set of n demands, we have seen in Section 1.1.1 that the dilation and the congestion provide two lower bounds for the routing time. This proves that the *dilation + congestion* of a paths system used for the routing procedure is a lower bound of twice the routing time. In a celebrated paper, Leighton, Maggs and Rao proved the following theorem :

Theorem 1.1 ([29]) *For any set of requests and a path system for these requests, there is an* **off-line** *routing protocol that needs $\mathcal{O}(C+D)$ steps to route all the requests, where C is the congestion and D is the dilation of the path system.*

In addition, in [49] the authors show that, given the set of packets to be sent, it is possible to find in polynomial time a path system with $C + D$ within a factor 4 of the optimum. Thus, Theorem 1.1 can be announced in a more general way :

Theorem 1.2 ([49]) *For any set of requests, there is an* **off-line** *routing protocol that needs $\mathcal{O}(C+D)$ steps to route all the requests, where $C + D$ is the minimum congestion + dilation over all the possible path systems.*

Furthermore, this routing protocol uses fixed buffer size, i.e. the queue size at all nodes is bounded by a constant at each step. Nevertheless, it is important to notice that a huge constant may be hidden inside the $\mathcal{O}$ notation. As we have said in the introduction, this result was further improved in [28] where the same authors gave an explicit algorithm. These algorithms to compute the optimal schedule are centralized. In a distributed algorithm nodes must make their decisions independently, based on the packets they see, without the use of a centralized scheduler. Then in [38] Ostrovsky and Rabani gave a distributed randomized algorithm running in $\mathcal{O}(C + D + \log^{1+\epsilon}(n))$ steps.

We refer to Scheideler's thesis [45] for a complete compilation of general packet routing algorithms.

1.1.3 On-line routing

In the on-line setting, the oldest on-line protocol that deviates only by a factor logarithmic in n from the best possible runtime $\mathcal{O}(C+D)$ for arbitrary path-collections is the protocol presented by Leighton, Maggs and Rao in the same paper [29], running in $\mathcal{O}(C + D\log(Dn))$ steps with high probability. The authors call the algorithm on-line, rather than distributed. This schedule assumes that the paths are given a priori, hence it does not focus on the problem of choosing the paths to route the packets.

The results of [1] provide a routing algorithm that is $\log n$ competitive with respect to the congestion. In other words, it is worse than an optimal off-line algorithm only by a factor $\log n$. In this setting the demands arrive one by one and the algorithm routes calls based on the current congestion on the various links in the network, so this can be achieved only via centralized control and serializing the routing requests. In [3] the authors gave a distributed algorithm that repeatedly scans the network so as to choose the routes. This algorithm requires shared variables on the edges of the network and hence is hard to implement. Note that the two on-line algorithms above depend on the demands and are therefore *adaptive*. Recall that an *oblivious* routing strategy is specified by a path system $\mathcal{P}$ and a function w assigning a weight to every path in $\mathcal{P}$. This function w has the property that for every source-destination pair (s,t), the system of flow paths $\mathcal{P}_{s,t}$ for (s,t) fulfills $\sum_{q\in\mathcal{P}_{s,t}} w(q) = 1$. One can think of this function as a frequency distribution among several paths going from an origin s to a destination t. In *adaptive* routing, however, the path taken by a packet may also depend on other packets or events taking place in the network during its travel. Remark that every oblivious routing strategy is obviously on-line and distributed.

The first paper to perform a worst case theoretical analysis on oblivious routing is the paper of Valiant and Brebner [54], who considered routing on specific network topologies such as the hypercube. They give a randomized oblivious routing algorithm. Borodin and Hopcroft [6] and subsequently [22] have shown that deterministic oblivious routing algorithms cannot approximate well the minimal load on any non-trivial network.

In a recent paper, Räcke [40] gave the construction of a polylog competitive oblivious routing algorithm for general undirected networks. It seems truly surprising that one can come close to minimal congestion without any information on the current load in the network. This result has been improved in [4]. Lower bounds on the competitive ratio of oblivious routing have been studied for various types of

networks. For example, for the d-dimensional mesh, Maggs et al. [35] gave the $\omega(\frac{C_*}{d}(\log n))$ lower bound on the competitive ratio of an oblivious algorithm on the mesh, where C_* is the optimal congestion.

So far, the oblivious algorithms studied in the literature have focused on minimizing the congestion while ignoring the dilation. In fact, the quality of the paths should be determined by the congestion C, and the dilation D. An open question is whether C and D can be controlled simultaneously. An appropriate parameter to capture how good is the dilation of a path system is the *stretch*, defined as the maximum over all packets of the ratio between the length of the path taken by the routing protocol and the length of a shortest path from source to destination. In a recent work, Bush et al. [8] considered again the case of the d-dimensional mesh. They presented an on-line algorithm for which C and D are both within $\mathcal{O}(d^2)$ of the potential optimal, i.e. $D = \mathcal{O}(d^2 D_*)$ and $C = \mathcal{O}(dC_* \log(n))$, where D_* is the optimal dilation (remark that by [35], it is impossible to have a factor better than $\omega(\frac{C_*}{d}(\log n))$).

There is a simple counter-example network that shows that in general the two metrics (*dilation* and *congestion*) are orthogonal to each other : take an adjacent pair of nodes u, v and $\Theta(\sqrt{n})$ disjoint paths of length $\Theta(\sqrt{n})$ between u and v. For packets traveling from u to v, any routing algorithm that minimizes congestion has to use all the paths, however, in this way some packets follow long paths, giving high stretch. Nevertheless, in grids [8], and in some special kind of geometric networks [7] the congestion is within a poly-logarithmic factor from optimal and stretch is constant (d the dimension). As mentioned before an interesting open problem is to find other classes of networks where the congestion and stretch are minimized simultaneously [2]. Possible candidates for such networks could be for example bounded-growth networks, or networks whose nodes are uniformly distributed in closed polygons, which describe interesting cases of wireless networks.

The recent paper of Maggs [34] surveys a collection of theoretical results that relate the congestion and dilation of the paths taken by a set of packets in a network to the time required for their delivery.

1.2 Routing Problems

The initial and final positioning of the packets has a direct influence on the time needed for their routing. Considering static packet configuration, the most studied constraints refer to the maximum number of packets that a node can send and receive. Due to their practical importance, some of these problems have specific names :

1. **Permutation routing** : each node is the origin and the destination of at most one packet. To measure the routing capability of an interconnection network, the partial permutation routing (PPR) problem is usually used as the metric.
2. (ℓ, k)**-routing :** each node is the origin of at most ℓ packets and destination of at most k packets. Permutation routing corresponds to the case $\ell = k = 1$ of (ℓ, k)-routing. Another important particular case is the $(1, k)$**-routing**, in which each node can send at most one packet and receive at most k packets.
3. $(1, any)$**-routing** : each node is origin of at most one packet but there are no constraints on the number of packets that a node can receive.
4. r**-central routing** : all nodes at distance at most r of a central node send one message to this central node.

In all these problems we are given an initial packet configuration, and the objective is to route all packets to their respective destinations minimizing the total routing time, under the constraint that each edge can be used by at most one packet at the same time.

Besides of the constraints about the initial and final positions of the packets, there also exist different routing models at the intermediate nodes of the network. For instance, in the *hot potato model* no packet can be stored at the nodes of the network, whereas in the *store-and-forward* at each step a packet can either stay at a node or move to an adjacent node. Another widely used model is the *wormhole* routing.

On the other hand, one can consider constraints on the number of incident edges that each node of the network can use to send or receive packets at the same time. In the Δ-port model [16], each node can send or receive packets through all its incident edges at the same time.

In this article we study the store-and-forward Δ-port model. In addition, we suppose that cohabitation of multiple packets at the same node is allowed. I.e. a queue is required for each outgoing edge at each node.

The nature of the links of the network is another factor that influences the routing efficiency. The type of links is usually one of the following : full-duplex or half-duplex. In the full-duplex case there are two links between two adjacent nodes, one in each direction. Hence two packets can transit, one in each direction, simultaneously. In the half-duplex case only one packet can transit between two nodes, either in one direction of the edge or in the other. In this paper we focus on both half and full-duplex links.

1.3 Topologies

We now give a brief summary of various cases of (ℓ, k)-routing and $(1, any)$-routing that have been studied for several specific topologies. More precisely, in Section 1.3.1 we list the most important results for some networks which have attracted a great interest in the literature, like hypercubes and circulant graphs. Then we move to plane grids in Section 1.3.2. It is well known that there exist only three possible tessellations of the plane into regular polygons [55] : squares, triangles and hexagons. These graphs are those which we study in this article.

1.3.1 Different network topologies

In [20] the authors studied the permutation routing problem in low-dimensional hypercubes ($d \leq 12$). They gave optimal or good in the worst case oblivious algorithms, i.e. algorithms in which the path used by a packet is entirely determined by its origin and its destination. An other network widely studied in the literature is the two dimensional mesh with row and column buses. This network can also be diversified according to the capacities of the buses. In [51] Suel gave a deterministic algorithm to solve the permutation routing problem in such networks. It gives a schedule using at most $n + o(n)$ steps and a queue of size 2, where the queue is the maximum number of packets that have to be stored at an intermediate node. He also proposed a deterministic algorithm for r-dimensional arrays with buses working in $(2 - \frac{1}{r})n + o(n)$ steps and still using queues of size 2. In [27], the authors studied the (ℓ, ℓ)-routing problem in the mesh grid with two diagonals and gave, for $\ell \geq 9$ a deterministic algorithm using $\frac{2\ell n}{9} + (\ell n^{2/3})$ steps.

In [19], the authors introduced an algorithm called *big foot* algorithm. The idea of this algorithm is to identify two types of links and to move towards the destination using first the links of the first type and then those of the second type. The algorithms we develop will use such a strategy. They give an optimal centralized algorithm for the permutation routing problem in full-duplex 2-circulant graphs and double-loop networks (i.e. oriented 2-circulant graphs).

Another network of great practical importance is the *double-loop* network : a network modeled by a graph with vertex set $v = \{v_0, \ldots, v_{n-1}\}$ such that there are two integers h_1 and h_2 such that $E = \{v_i v_{i\pm h_1}, v_i v_{i\pm h_2}\}$. The permutation routing problem in this network is studied in [14]. The authors gave an algorithm for the permutation routing problem which in mean uses 1.12ℓ steps (the mean being empirically measured). In [15] the authors described an optimal centralized permutation routing algorithm in k-circulant graphs ($k \geq 2$), and in [42] an optimal distributed permutation routing in 2-circulant graphs was obtained.

The problem has been also studied for packets arriving dynamically. In [18], the author gave an optimal online schedule for the linear array. He also gave a 2-approximation for rings and show that, using shortest path routing, no better approximation algorithm exists. In [21], the authors studied Cube Connected Cycles : $CCC(n, 2^n)$ (hypercubes of dimension n where each node is replaced by a cycle of

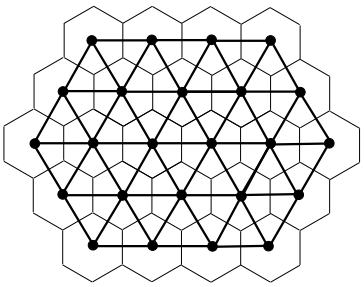

FIGURE 1 – Hexagonal network (△) and hexagonal tessellation (⬡).

length n). They gave an algorithm working in $\mathcal{O}(n^2)$ with $\mathcal{O}(1)$ buffers for the online partial permutation routing (PPR).

1.3.2 Plane grids

Maybe the most studied networks in the literature are the two dimensional grids (or plane grids), and among them in particular the square grid has deserved special attention. Let us briefly overview what has been previously done on (ℓ, k)-routing in plane grids.

In [32] the first optimal permutation routing (with running time $2n-2$), and queues of size 1008 appears. The queue size is reduced in [41] to 112 and further in [48] to 81. Furthermore, in [48] the authors provide another algorithm running in near-optimal time $2n + \mathcal{O}(1)$ steps with a maximum queue size of only 12. [36] gives an *asymptotically* optimal algorithm for $(1, k)$-routing on plane grids, with queues of small constant size. They introduced for the first time the $(1, k)$-routing and the $(1, any)$-routing problems. This result was further improved in [47], where the authors gave near-optimal deterministic algorithm running in $\sqrt{k}\frac{n}{2} + \mathcal{O}(n)$ steps. They gave another algorithm, slightly worse in terms of number of steps, but with queues of size only 3. They also studied the general problem of (ℓ, k)-routing in square grids. They proposed lower bounds and near-optimal randomized and deterministic algorithms. They finally extended them to higher dimensional meshes. They performed (ℓ, ℓ)-routing in $\mathcal{O}(\ell n)$ steps, the lower bound being $\Omega(\sqrt{\ell k n})$ for (ℓ, k)-routing. Finally, in [39], the authors gave deterministic and randomized algorithms for (ℓ, k)-routing in square grids, with constant queue size. The running time is $\mathcal{O}(\sqrt{\ell k n})$ steps, which is optimal according to the bound of [47]. This work closed a gap in the literature, since optimal algorithms were only known for $\ell = 1$ and $\ell = k$.

Nodes in a hexagonal network are placed at the vertices of a regular triangular tessellation, so that each node has up to six neighbors. In other words, a hexagonal network is a finite subgraph of the triangular grids. These networks have been studied in a variety of contexts, specially in wireless and interconnection networks. The most known application may be to model cellular networks with hexagonal networks where nodes are base stations. But these networks have been also applied in chemistry to model benzenoid hydrocarbons [52, 25], in image processing and computer graphics [26].

In a radiocommunication wireless environment [37], the interconnection network among base stations constitutes a hexagonal network, i.e. a triangular grid, as it is shown in FIG. 1.

Tessellation of the plane with hexagons may be considered as the most natural because cells have optimal diameter to area ratio. Hexagonal networks are finite subgraphs of the triangular grid. The triangular grid can also be obtained from the basic 4-mesh by adding NE to SW edges, which is called a 6-mesh in [53]. Here we study convex subgraphs, i.e. that contain all shortest paths between all pairs of nodes, of the square, triangular and hexagonal grid. Summarizing, to the best of our knowledge the only optimal algorithms concerning (ℓ, k)-routing on plane grids (according to the lower bound of [47]) have been found on square grids, but modulo a constant factor [39]. On triangular and hexagonal grids, the best results are randomized algorithms with good performance [46].

1.4 Our Contribution

In this paper we study the permutation routing, r-central and (ℓ, k)-routing problems on plane grids, that is square grids, triangular grids and hexagonal grids. We use the *store-and-forward* Δ-port model, and we consider both full and half-duplex networks.

We have seen in Section 1.3.2 that the only plane grid for which there existed an optimal (ℓ, k)-routing is the square grid. In addition, what is important is that the results of these articles concerning (ℓ, k)-routing in plane grids are optimal modulo a constant factor. In this paper we improve these results by giving tight algorithms including the constant, in the cases of square, triangular and hexagonal grids. It is important to stress that all the algorithms presented in this paper are **distributed** (except the one given in Appendix B), i.e. can be implemented independently at each node. Our algorithms only use shortest paths, therefore they achieve minimum stretch. In addition, the algorithms are oblivious, so they can be used in an on-line scenario, unless the performance guarantees that we prove apply only to the off-line case. The main new results of this article are the following :

1. First tight (also including the constant factor) permutation routing algorithms in full-duplex hexagonal grids, and half duplex triangular and hexagonal grids.
2. First tight (also including the constant factor) r-central routing algorithms in triangular and hexagonal grids.
3. First tight (also including the constant factor) (k, k)-routing algorithms in square, triangular and hexagonal grids.
4. Good approximation algorithms for (ℓ, k)-routing in square, triangular and hexagonal grids.

This paper is structured as follows. In Section 2 we study the permutation routing problem. Although permutation routing had already been solved for square grids, we begin in Section 2.1 by illustrating our algorithm for such grids. Then in Section 2.2 we give a tight permutation routing algorithm for half-duplex triangular grids, using the optimal algorithm of [44]. In Section 2.3 we provide a tight permutation routing algorithm for full-duplex hexagonal grids and a tight permutation routing algorithm for half-duplex hexagonal grids. In Section 3 we focus on $(1, any)$-routing, giving an optimal r-central routing algorithms for the three types of grids. We finally move in Section 4 to the general (ℓ, k)-routing problem. We provide a distributed algorithm for (ℓ, k)-routing in any grid, using the ideas of the optimal algorithm for permutation routing. We also prove lower bounds for the worst-case running time of any algorithm using shortest path routing. In addition, these lower bounds allow us to prove that our algorithm turns out to be tight when $\ell = k$, yielding in this way a tight (k, k)-routing algorithm in square, triangular and hexagonal grids. We also propose in Appendix B an approach to (ℓ, k)-routing in terms of a graph coloring problem : the Weighted Bipartite Edge Coloring. We give a centralized algorithm using this reduction.

2 Permutation Routing

As we have already said in Section 1, in the permutation routing problem, each processor is the origin of at most one packet and the destination of no more than one packet. The goal is to minimize the number of time steps required to route all packets to their respective destinations. It corresponds to the case $\ell = k = 1$ of the general (ℓ, k)-routing problem. This problem has been studied in a wide diversity of scenarios, such as Mobile Ad Hoc Networks [23], Cube-Connected Cycle (CCC) Networks [21], Wireless and Radio Networks [10], All-Optical Networks [33] and Reconfigurable Meshes [9].

In a grid with full-duplex links an edge can be crossed simultaneously by two messages, one in each direction. Equivalently, each edge between two nodes u and v is made of two independent arcs $\{uv\}$ and $\{vu\}$, as illustrated in Fig. 2a.

Remark 2.1 *If the network is half-duplex, it is easy to construct a 2-approximation algorithm from an optimal algorithm for the full-duplex case by introducing* odd-even *steps, as explained for example in*

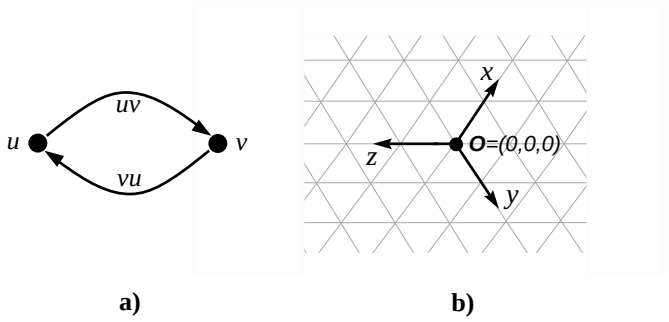

FIGURE 2 – **a)** Each edge consists of two independent links. **b)** Axis used in a triangular grid.

[14].

2.1 Square grid

Many communication networks are represented by graphs satisfying the following property : for any pair of nodes u and v, the edges of a shortest path from u to v can be partitioned into k disjoint classes according to a well-defined criterium. For instance, on a triangular grid the edges of a shortest path can be partitioned into *positive* and *negative* ones [44]. Similarly, on a k-circulant graph the edges can be partitioned into k classes according to their length.

In graphs that satisfy this property there exists a natural routing algorithm : route all packets along one class of edges after another. For hexagonal networks this algorithm turns out to be optimal [44]. Optimality for 2-circulant graphs is proved using a static approach in [19], and recently using a dynamic distributed algorithm in [42]. In [19] the authors introduce the notion of *big-foot* algorithms because their algorithm routes packets first along *long hops* and then along *short hops* in a 2-circulant graph.

On the square grid, the *big-foot* algorithm consists of two phases, moving each packet first horizontally and then vertically. In this way a packet may wait only during the second phase. Using that all destinations are distinct, the optimality for square grid is easy to prove. Summarizing, it can be proved that

Theorem 2.1 *There is a translation invariant oblivious optimal permutation routing algorithm for full-duplex networks that are convex subgraphs of the infinite square grid.*

2.1.1 Regarding the queue size

Of course, this is not the first optimal permutation routing result on square grids, as the classical $x-y$ routing (first route packets through the horizontal axis, and then through the vertical axis) has been used for a long time. Thus, another more challenging issue is to reduce the queue size, as we have already discussed in Section 1.3.2. Leighton describes in [31] a simple off-line algorithm for solving any permutation routing problem in $3n-3$ steps on a $n \times n$ square grid, using queues of size one. Since the diameter of a $n \times n$ square grid is $2n-2$, this algorithm provides a $\frac{3}{2}$-approximation. The main drawback is that this algorithm is off-line and centralized. In contrast, our oblivious distributed algorithm is optimal in terms of running time, but it is easy to see that on a $n \times n$ square grid, the queue size can be $\frac{n-1}{2}$. Up to date, the best algorithm running in optimal time to route permutation routing instances on square grids is the algorithm of Sibeyn et al. [48], using queues of size 81. So far, there is no algorithm that guarantees optimal running time with queues of size 1, and it is unlikely that such an algorithm exists.

Remark 2.2 *The same observation regarding the unbounded queue size applies to all the algorithms described in this article. However, our aim is to match the optimal running time, rather than minimizing the queue size. Additionally, it turns out that some appropriate modifications of the permutation routing algorithms that we provide for plane grids allow us to find oblivious algorithms which route any permutation within a factor 3 of the optimal running time, and using queues of size 1 (in fact, we can say something stronger : we just need memory to keep 1 message at each node). We do not describe these modifications in this article.*

2.2 Triangular grid

We use the addressing scheme introduced in [37] and used also in [44] : we represent any address on a basis consisting of three unitary vectors $\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$ on the directions of three axis x, y, z with a 120 degree angle among them, intersecting on an arbitrary (but fixed) node O . This node is the origin and is given the address $\boldsymbol{O} = (0, 0, 0)$. This basis is represented in Fig. 2b. Thus, we can assume that each node $P \in V$ is labeled with an address $\boldsymbol{P} = (P_1, P_2, P_3)$ expressed in this basis $\{\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}\}$ with respect to the origin O. At the beginning, each node S knows the address of the destination node D of the message placed initially at S, and computes the relative address $\overrightarrow{SD} = \boldsymbol{D} - \boldsymbol{S}$ of the message. Note that this relative address does not depend on the choice of the origin node O. This relative address is the only information that is added in the heading of the message to be transmitted, constituting in this way the packet to be sent through the network.

Using that $\boldsymbol{i} + \boldsymbol{j} + \boldsymbol{k} = 0$, it is easy to see that if (a, b, c) and (a', b', c') are the relative addresses of two packets, then $(a, b, c) = (a', b', c')$ if and only if there exists $d \in \mathbb{Z}$ such that $a' = a + d$, $b' = b + d$, and $c' = c + d$.

We say that an address $\overrightarrow{SD} = (a, b, c)$ is of the *shortest path form* if there is a path from node S to node D, consisting of a units of vector $\boldsymbol{i}$, b units of vector $\boldsymbol{j}$ and c units of vector $\boldsymbol{k}$, and this path has the shortest length.

Theorem 2.2 ([37]) *An address (a, b, c) is of the shortest path form if and only if at least one component is zero, and any two components do not have the same sign.*

Corollary 2.1 ([37]) *Any address has a unique shortest path form.*

Thus, each address $\overrightarrow{SD}$ written in the shortest path form has at most two non-zero components, and they have different sign. In fact, it is easy to find the shortest path form using the next result.

Theorem 2.3 ([37]) *If $\overrightarrow{SD} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$, then*

$$|\overrightarrow{SD}| = \min(|a - c| + |b - c|, |a - b| + |b - c|, |a - b| + |a - c|).$$

Permutation routing on full-duplex triangular grids has been solved recently [44] attaining the distance lower bound of $\ell_{\max}$ routing steps, where $\ell_{\max}$ is the maximum length over the shortest paths of all packets to be sent through the network.

As said in Remark 2.1, ff the network is half-duplex, one can construct a 2-approximation algorithm from an optimal algorithm for the full-duplex case by introducing *odd-even* steps. Thus, using this algorithm we obtain an upper bound of $2\ell_{\max}$ for half-duplex triangular grids.

Let us show with an example that this naïve algorithm is tight. That is, we shall give an instance requiring at least $2\ell_{\max}$ running steps, implying that no better algorithm for a general instance exists. Indeed, consider a set of nodes distributed along a line on the triangular grid. We fix $\ell_{\max}$ and an edge e on this line, and put $\ell_{\max}$ packets at each side of e along the line, at distance at most $\ell_{\max} - 1$ from an end-vertex of e. For each packet, each destination is placed at the other side of e with respect to its origin, at distance exactly $\ell_{\max}$ from the origin. It is easy to check that the congestion of e (that is, the number of shortest paths containing e) is $2\ell_{\max}$, and thus any algorithm using shortest path routing

cannot perform in less than $2\ell_{\max}$ steps. On the other hand, $\ell_{\max}$ is a lower bound for any distance, yielding that the approximation ratio of our algorithm is at most 2.

Previous observations allow us to state the next result :

Theorem 2.4 *There exists a tight permutation routing algorithm for half-duplex triangular grids performing in at most $2\ell_{\max}$ steps, where $\ell_{\max}$ is the maximum length over the shortest paths of all packets to be sent. This algorithm is a 2-approximation algorithm for a general instance.*

2.3 Hexagonal grid

In a hexagonal grid one can define three types of *zigzag chains* [50], represented with thick lines in FIG. 3. Similarly to the triangular grid, in the hexagonal grid any shortest path between two nodes uses at most two types of zigzag chains [50]. Let us now give a lower bound for the running time of any algorithm. Consider the edge labeled as e in FIG. 3, and the two chains containing it (those shaping an X). Fix $\ell_{\max}$ and e, and put one message on all nodes placed at both chains at distance at most $\ell_{\max}-1$ from an endvertex of e. As in the case of the triangular grid, choose the destinations to be placed on the other side of e along the same zigzag chain than the originating node, at distance exactly $\ell_{\max}$ from it. It is clear that all the shortest paths contain e. It is also easy to check that the congestion of e is in this case $4\ell_{\max}-4$, constituted of symmetric loads $2\ell_{\max}-2$ in each direction of e. Thus, $2\ell_{\max}-2$ establishes a lower bound for the running time of any algorithm in the full-duplex case, whereas $4\ell_{\max}-4$ is a lower bound for the half-duplex case, under the assumption of shortest path routing.

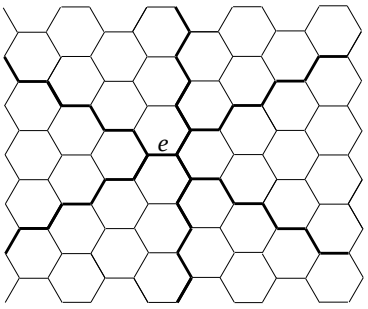

FIGURE 3 – Zigzag chains in a hexagonal grid.

Let us now describe a routing algorithm which reaches this bound. We have three types of edges according to the angle that they form with any fixed edge. Each edge belongs to exactly two different chains, and conversely each chain is made of two types of edges. Moreover, in an infinite hexagonal grid any 2 chains of different type intersect exactly on one edge.

Given a pair of origin and destination nodes S and D, it is possible to express the relative address $D-S$ counting the number of steps used by a shortest paths on each type of chain. In this way we obtain an address $D-S=(a,b,c)$ on a generating system made of unitary vectors following the directions of the three types of chains (it is not a basis in the strict sense, since these vectors are not linearly independent on the plane. However, we will call it so). Choose this basis so that the three vectors form angles of 120 degrees among them. As it happens on the triangular grid [37, 43], there are at most two non-zero components (see [50]), and in that case they must have different sign. Nevertheless, now the address is not unique, since an edge placed at the *bent* (that is, a change from a type of chain to another) of a shortest path is part of both types of chains. Anyway, this ambiguity is not a problem in the algorithm that propose, as we will see below.

Suppose first that edges are bidirectional or, said otherwise, full-duplex. Roughly, the idea is to use the optimal algorithm for triangular grids described in [43], and adapt it to hexagonal grids. For that purpose we label the three types of zigzag chains c_1, c_2, c_3, and the three types of edges e_1, e_2, e_3. Without

loss of generality, we label them in such a way that c_1 uses edges of type e_2 and e_3, c_2 uses e_1 and e_3, and c_3 uses e_1 and e_2 (see FIG. 4).

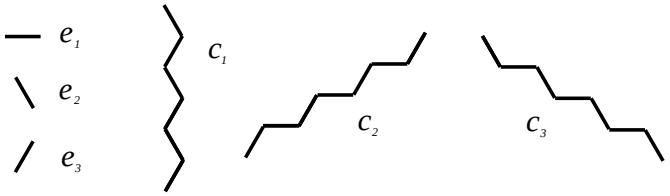

FIGURE 4 – 3 types of chains and edges in a hexagonal grid.

For each type of edge, we define two phases according to the type of chain that uses this type of edge. This defines two global phases, namely : during Phase 1, c_1 uses e_2, c_2 uses e_3, and c_3 uses e_1. Conversely, during Phase 2 c_1 uses e_3, c_2 uses e_1, and c_3 uses e_2.

We suppose that at each node packets are grouped into distinct queues according to the next edge (according to the rules of the algorithm) along its shortest path. Given the relative addresses $D - S$ in the form (a, b, c), the algorithm can be described as follows.

At each node u of the network :

1) During the first step, move all packets along the direction of their negative component. If a packet's address has only a positive component, move it along this direction.

2) From now on, change alternatively between Phase 1 and Phase 2. At each step (the same for both phases) :

a) If there are packets with negative components, send them immediately along the direction of this component.

b) If not, for each outgoing edge order the packets in decreasing number of remaining steps, and send the first packet of each queue.

3) If at some point, all the packets in u have remaining distance one, send them immediately.

Let us analyze the correctness and optimality of this algorithm.

In **1)** all packets can move, since initially there is at most one packet at each node. In **2a)**, there can only be one packet with negative component at each outgoing edge [43]. In **2b)** *the* packet with maximum remaining length at each outgoing edge is unique, since all these packets are moving along their last direction (their negative component is already finished, otherwise they would be in **2a)**) and each node is the destination of at most one packet. Hence, using this algorithm every 2 steps (one of Phase 1 and one of Phase 2) the maximum remaining distance over all packets decreases by one. Moreover, during the first step all packets decrease their remaining distance by one. Because of this, after the $(2\ell_{\max} - 3)th$ step the maximum remaining distance has decreased at least $1 + \frac{2\ell_{\max}-4}{2} = \ell_{\max} - 1$ times, hence the maximum remaining distance is 1, and we are in **3)**. Since all destinations are different, all packets can reach simultaneously their destinations. Thus, the total running time is at most $2\ell_{\max} - 3 + 1 = 2\ell_{\max} - 2$, meeting the *worst case* lower bound. Again, $\ell_{\max}$ is a lower bound for any instance, hence the algorithm constitutes a 2-approximation for a general instance.

Theorem 2.5 *There exists a tight permutation routing algorithm for full-duplex hexagonal grids performing in $2\ell_{\max} - 2$ steps, where $\ell_{\max}$ is the maximum length over the shortest paths of all packets to be sent.*

Remark 2.3 *The optimality stated in Theorem 2.5 is true only under the assumption of shortest path routing. This means that for certain traffic instances the total deliver time may be shorter if some packets do not go through their shortest path. To illustrate this phenomenon, consider the example of* FIG. *5.*

Node labeled i wants to send a message to node labeled i', for $i = 1, \ldots, 8$. We have that $\ell_{\max} = 5$, and thus our algorithm performs in $2\ell_{\max} - 2 = 8$ steps. It is clear that all shortest paths use edge e, and its congestion bottlenecks the running time. Suppose now that we route the messages originating at even nodes through the path defined by the edges $\{abcd\}$, instead of $\{fe\}$, and keep the shortest path routing for messages originating at odd nodes. One can check that with this routing only 7 steps are required.

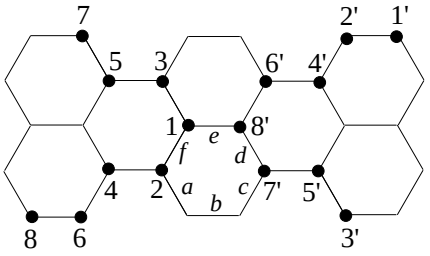

FIGURE 5 – Shortest path is not always the best choice.

In the half-duplex case, just introduce again odd-even steps in both phases. Thus, we have Phase 1-even, Phase 1-odd, Phase 2-even, and Phase 2-odd, which take place sequentially. Now, **1)** consists obviously of two steps (even/odd). Using this algorithm, every 4 steps the maximum remaining distance decreases by one. In addition, during the first 2 steps and during the last 2 steps all packets decrease their remaining distance by one. Thus, the total running time is at most twice the time of the full-duplex case, that is $2(2\ell_{\max} - 2) = 4\ell_{\max} - 4$ steps, meeting again the lower bound for the running time of any routing algorithm using shortest path routing. Again, this algorithm constitutes a 4-approximation for a general instance.

Theorem 2.6 *There exists an tight permutation routing algorithm for half-duplex hexagonal grids performing in $4\ell_{\max} - 4$ steps, where $\ell_{\max}$ is the maximum length over the shortest paths of all packets to be sent.*

Remark 2.4 *As explained in Appendix A, there exists an embedding of the triangular grid into the hexagonal grid with load, dilation, and congestion 2. Using this embedding, any algorithm performing on k steps on the triangular gird performs on $2k$ steps on the hexagonal grid. Using this fact, we obtain a permutation routing algorithm on full-duplex hexagonal grids performing on $2\ell_{\max}$ steps. Note that the optimal result given in Theorem 2.5 is slightly better.*

The same applies to half-duplex hexagonal networks, with a running time of $4\ell_{\max}$ using the embedding, in comparison to $4\ell_{\max} - 4$ steps given by Theorem 2.6.

3 $(1, any)$-Routing

In this case the routing model is the following : each packet has at most one packet to send, but there are no constraints on the destination. That is, in the worst case all packets can be sent to one node. This special case where all packets want to send a message to the same node in often called *gathering* in the literature [5]. Notice that this routing model is conceptually different from the $(1, k)$-routing, where the maximum number of packets that a node can receive is fixed *a priori*.

Square grid Assume first that edges are bidirectional. The modifications for the half-duplex case are similar to those explained in the previous section.

We will focus on the case where all packets surrounding a given vertex want to send a packet to that vertex. We call this situation *central* routing, and if we want to specify that all nodes at distance at

most r from the center want to send a packet, we note it as *r-central* routing. Note that this situation is realistic in many practical applications, since the central vertex can play the role of a router or a gateway in a local network.

Lemma 3.1 (Lower Bound) *The number of steps required in a r-central routing is at least $\binom{r+1}{2}$.*

Proof: Let us use the bisection bound [17] to prove the result. It is easy to count the number of points at distance at most r from the center, which is $4\binom{r+1}{2}$. Now consider the cut consisting of the four edges outgoing from the central vertex. All packets must traverse one of these edges to arrive to the central vertex. This cut gives the bisection bound of $4\binom{r+1}{2}/4$ routing steps. □

Let us now describe an algorithm meeting the lower bound.

Proposition 3.1 *There exists an optimal r-central routing algorithm on square grids performing in $\binom{r+1}{2}$ routing steps.*

Proof: Express each node address in terms of the relative address with respect to the central vertex. In this way each node is given a label (a, b). Then, for each packet placed in a node with label (a, b) our routing algorithm performs the following :

- If $ab = 0$, send the packet along the direction of the non-zero component.
- If $ab > 0$, send the packet along the vertical axis.
- If $ab < 0$, send the packet along the horizontal axis.

Queues are managed so that the packets having greater remaining distance have priority.

This routing divides the square grid into 4 subregions surrounding the central vertex, as shown in Fig. 6. The type of routing performed in each subregion is symbolized by an arrow.

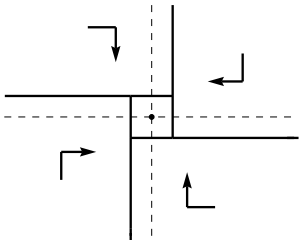

Figure 6 – Division of the grid in the proof of Proposition 3.1.

Let us now compute the running time in the r-central case. It is obvious that using this algorithm all packets are sent to the 4 axis outgoing from the central vertex. The congestion of the edge in the axis containing the central vertex along each line is $1 + 2 + 3 + \ldots + r = \binom{r+1}{2}$. Since at each step one packet reaches its destination along each line, we conclude that $\binom{r+1}{2}$ is the total running time of the algorithm. □

Triangular grid The same idea of the square grid applies to the triangular grid. In this case, the number of nodes at distance at most r is $6\binom{r+1}{2}$. The cut is made of 6 edges. Dividing the plane onto 6 subregions gives again an optimal algorithm performing in $\binom{r+1}{2}$ steps.

Hexagonal grid The same idea gives an optimal routing in the r-central case. In this case the degree of each vertex is 3, and then it is easy to check (maybe a drawing using Fig. 3 can help) that there are $3\binom{r+1}{2}$ nodes at distance at most d that may want to send a message to the central vertex, and the cut has size 3. As expected, the running time is again $\binom{r+1}{2}$.

4 (ℓ, k)-Routing

Recall that in the general (ℓ, k)-routing problem each node can send at most ℓ packets and receive at most k packets. We propose a distributed approximation algorithm using the ideas of the algorithms that we have developed for the permutation routing problem. We also provide lowers bounds for the running time of any algorithm using shortest path routing, that allow us to prove that our algorithm is tight when $\ell = k$, on any grid.

Remark 4.1 *We also propose in Appendix B an approach to find a solution of the (ℓ, k)-routing problem, on any grid, using the problem of* Weighted Edge Coloring *in a bipartite graph. Nevertheless, the algorithm obtained using this approach is centralized.*

We start by describing the results for full-duplex triangular grids. The results can be completely adapted to square grids, but we focus on the other grids since there were few results in the literature. We also show how to adapt the results to hexagonal grids and to the half-duplex version. In this section we denote $c := \left\lceil \frac{\max\{\ell,k\}}{\min\{\ell,k\}} \right\rceil = \lceil \max\{\frac{\ell}{k}, \frac{k}{\ell}\} \rceil$. Note that $c \geq 1$. Lemma 4.1 and Lemma 4.2 provide two lower bounds for the running time of any algorithm using shortest paths.

Lemma 4.1 (First lower bound) *The worst-case running time of any algorithm for (ℓ, k)-routing on full-duplex triangular grids using shortest path routing satisfies*

$$\textit{Running time} \geq \min\{\ell, k\} \cdot \ell_{\max}$$

Proof: Consider a set of $\ell_{\max}$ nodes placed along a line, placed consecutively at one side of a distinguished edge e. Each node wants to send $\min\{\ell, k\}$ messages to the nodes placed at the other side of e along the line, at distance $\ell_{\max}$ from it. Then the congestion of e is $\min\{\ell, k\} \cdot \ell_{\max}$, giving the bound. □

Definition 4.1 *Given a vertex v, we call the rectangle of side (a, b) starting at v the set $R^v_{a,b} = \{v + \alpha\mathbf{i} + \beta\mathbf{j}, 0 \leq \alpha < a, 0 \leq \beta < b\}$ We call such a rectangle a square if $a = b$. Notice that in the triangular grid the node set is generated by $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$, where $\mathbf{k} = -\mathbf{i} - \mathbf{j}$, as we have explained in Section 2.2.*

Using standard graph terminology, given a graph $G = (V, E)$ and a subset $S \subseteq V$, the set $\Gamma(S)$ denotes the (open) neighborhood in G of the vertices in S. The following theorem can be found, for example, in [13].

Theorem 4.1 (Corollary of Hall's theorem [13]) *Let $G = (V, E)$ be a bipartite graph, with $V = X \cup Y$. If for all subsets A of X, $|\Gamma(A)| \geq c|A|$, then for each $x \in X$, there exists $S_x \subset Y$ such that $|S_x| = c$, and $\forall x, x' \in X$, $S_x \cap S_{x'} = \emptyset$ and $S_x \subset \Gamma(x)$.*

We use this theorem to prove the following lower bound.

Lemma 4.2 (Second lower bound) *The worst-case running time of any algorithm for (ℓ, k)-routing on full-duplex triangular grids using shortest path routing satisfies*

$$\textit{Running time} \geq \left\lceil \frac{\max\{\ell, k\}}{4} \cdot \left\lfloor \frac{\ell_{\max} + 1}{\sqrt{c+1}} \right\rfloor \right\rceil \ ,$$

where $c = \left\lceil \frac{\max\{\ell,k\}}{\min\{\ell,k\}} \right\rceil$.

Proof: Suppose without loss of generality that $\ell \geq k$, otherwise replace ℓ by $\max\{\ell, k\}$. Let v be a vertex, and consider the square $R^v_{d,d}$, with $d := \left\lfloor \frac{\ell_{\max}+1}{\sqrt{c}+1} \right\rfloor$. We claim that all nodes inside this square can send ℓ messages such that all destination nodes are in the destination set $D = R^v_{d+\ell_{\max},d+\ell_{\max}} \setminus R^v_{d,d}$. Let S be the subgrid generated by positive linear combinations of the vectors $\mathbf{i}$ and $\mathbf{j}$. More precisely, $S := \{v + \alpha\boldsymbol{i} + \beta\boldsymbol{j}, \alpha \geq 0, \beta \geq 0\}$. FIG. 2b gives a graphical illustration.

To prove this, we consider a bipartite graph H on vertex set $R^v_{d,d} \cup D$, with an edge between a vertex of $R^v_{d,d}$ and a vertex of D if they are at distance at most $\ell_{\max}$ in S. To apply Theorem 4.1, we have to show that any subset of vertices $A \subset R^v_{d,d}$ has at least $c|A|$ neighbors in H. Theorem 4.1 will then ensure the existence of a feasible repartition of the messages from vertices of $R^v_{d,d}$ to those of D such that they all travel a distance at most $\ell_{\max}$.

Given $A \subset R^v_{d,d}$, let us call $D_A := \{u \in D \ : \mathrm{dist}_S(A,u) \leq \ell_{\max}\}$, where $\mathrm{dist}_S(A,u)$ means the minimum distance in S from any vertex of A to the vertex u. For any $A \subset R^v_{d,d}$, we need to show that

$$|D_A| \geq c|A| \tag{1}$$

Without loss of generality we suppose that A is maximal, in the sense that there is no set A' strictly containing A with $D_A = D_{A'}$. Instead of considering all possible sets A, we will show below that we can restrict ourselves to rectangles. Hence given a set A, we denote by R_A the smallest rectangle containing the subset of vertices A. We first claim that

$$|D_{R_A} \setminus D_A| \leq |R_A \setminus A| \tag{2}$$

Indeed, this equality can be shown by induction on $|R_A \setminus A|$. For $|R_A \setminus A| = 0$ the equality is trivial. Suppose that it is true for $|R_A \setminus A|$. The induction step consists in showing that there is an element x in $R_A \setminus A$ such that $|D_{R_A} \setminus D_{A\cup\{x\}}| - |D_{R_A} \setminus D_A| \leq 1$ (note that $D_{R_{A\cup\{x\}}} = D_{R_A}$) :

- If there exists x such that $x + \boldsymbol{j}$ and $x - \boldsymbol{i}$ are in A and $x - \boldsymbol{j}$ is not in A, then we select this x. From x the only new vertex we may add to D_A is $x + \ell_{\max}\boldsymbol{i}$.
- Otherwise, if there exists x such that $x - \boldsymbol{j}$ and $x - \boldsymbol{i}$ are in A and $x + \boldsymbol{i}$ is not in A, then we select this x. In this case the only new vertex we may add to D_A is $x - \ell_{\max}\boldsymbol{k}$.
- If none of the previous cases holds, since R_A is the smallest rectangle containing A, and A is maximal, then necessarily there exists an x such that $x + \boldsymbol{i}$ and $x - \boldsymbol{j}$ are in A and $x - \boldsymbol{i}$ is not in A. We select this x, and the only new vertex we may add to D_A is $x + \ell_{\max}\boldsymbol{j}$.

Thus, in all cases there exists an x adding at most one neighbor to $D_{R_A} \setminus D_A$, which finishes the induction step and proves Equation (2). To finish the proof of the fact that we can restrict ourselves to rectangles, we show that, for any subset A, if Inequality (1) holds for R_A, then it also holds for A. Indeed, Inequality (1) applied to R_A gives :

$$c|R_A| \leq |D_{R_A}| \ , \quad \text{which is equivalent to}$$

$$c(|A| + |R_A \setminus A|) \leq |D_A| + |D_{R_A} \setminus D_A| \tag{3}$$

Using Inequality (2) and the fact that $c \geq 1$, Inequality (3) clearly implies that Inequality (1) holds.

Henceforth we assume that A is a rectangle. The last simplification consists in proving that we can restrict ourselves to rectangles containing v. In other words, it will be sufficient to prove Inequality (1) for all rectangles $R^v_{a,b}$. Given a rectangle R not positioned at v, the rectangle R' of the same size positioned at v has less neighbors, hence if Inequality (1) holds for R', it also holds for R.

Finally let us prove that Inequality (1) holds for all rectangles $R^v_{a,b}$, with $1 \leq a, b < d$. We have that $|R^v_{a,b}| = ab$ and $|D_{R^v_{a,b}}| = (a + \ell_{\max})(b + \ell_{\max}) - d^2$.

By the choice of d, starting from the Inequality $d^2 \leq \frac{(\ell_{\max}+1)^2}{c+1}$ and using that $1 \leq a, b$, one obtains that $d^2 c \leq (\ell_{\max} + a)(\ell_{\max} + b) - d^2$ for any $1 \leq a, b < d$. This implies, using $a, b < d$, that $cab \leq (a + \ell_{\max})(b + \ell_{\max}) - d^2$ for any $1 \leq a, b < d$, hence Inequality (1) (i.e. $c|R^v_{a,b}| \leq |D_{R^v_{a,b}}|$) holds.

So by Theorem 4.1, each one of the d^2 nodes in $R^v_{d,d}$ can send ℓ messages to the nodes of D. Since the number of edges going from $R^v_{d,d}$ to D is $4d - 1$, we apply the bisection bound discussed in Section 1.1.1 to conclude that there is an edge of the border of the square $R^v_{d,d}$ with congestion at least $\left\lceil \frac{\ell \cdot d^2}{4d-1} \right\rceil > \left\lceil \frac{\ell \cdot d}{4} \right\rceil$. This finishes the proof of the lemma. □

We observe that this second lower bound is strictly better than the first one if and only if

$$\frac{c}{\sqrt{c+1}} > \frac{4\ell_{\max}}{\ell_{\max} + 1}$$

If both c and $\ell_{\max}$ are big, the condition becomes approximately :

$$\frac{\max\{\ell, k\}}{\min\{\ell, k\}} > 16$$

That is, the second lower bound is better when the difference between ℓ and k is big. This is the case of broadcast or gathering, where messages are originated (or destined) from (or to) a small set of nodes of the network.

The two lower bounds can be combined to give :

Lemma 4.3 (Combined lower bound) *The worst-case running time of any algorithm for (ℓ, k)-routing on full-duplex triangular grids using shortest path routing satisfies*

$$\textit{Running time} \geq \max\left(\ell_{\max} \cdot \min\{\ell, k\},\ \max\{\ell, k\} \cdot \left\lfloor \frac{\ell_{\max} + 1}{4\sqrt{c+1}} \right\rfloor\right) \approx \ell_{\max} \cdot \max\left(\min\{\ell, k\},\ \frac{\max\{\ell, k\}}{4\sqrt{c+1}}\right)$$

Now we provide an algorithm from which we derive an upper bound.

Proposition 4.1 (Upper bound (algorithm)) *The algorithm for (ℓ, k)-routing on full-duplex triangular grids is the following : route all packets as in the permutation routing case. That is, at each node send packets first in their negative component, breaking ties arbitrarily (there can be ℓ packets in conflict in a negative component). If there are no packets with negative components, send any of the (at most k) packets with maximum remaining distance.*

$$\textit{Running time} \leq \begin{cases} \min\{\ell, k\} \cdot \frac{c(c-1)}{2} + \max\{\ell, k\} \cdot (\ell_{\max} - c + 1) & , \textit{ if } c \leq \ell_{\max} \\ \min\{\ell, k\} \cdot \frac{\ell_{\max}(\ell_{\max}+1)}{2} & , \textit{ if } c > \ell_{\max} \end{cases}$$

Proof: Suppose again without loss of generality that $\ell \geq k$. We proceed by decreasing induction on $\ell_{\max}$. We prove that after $\min\{\ell, \ell_{\max} k\}$ steps, each paquet will be at distance at most $\ell_{\max} - 1$ of its destination. This yields

$$\text{Running time}(\ell_{\max}) \leq \min\{\ell, \ell_{\max} k\} + \text{Running time}(\ell_{\max} - 1)$$

$$\leq \min\{\ell, \ell_{\max} k\} + \begin{cases} \min\{\ell, k\} \cdot \frac{c(c-1)}{2} + \max\{\ell, k\} \cdot (\ell_{\max} - c) & , \text{if } c \leq \ell_{\max} - 1 \\ \min\{\ell, k\} \cdot \frac{\ell_{\max}(\ell_{\max}-1)}{2} & , \text{if } c > \ell_{\max} - 1 \end{cases}$$

$$\leq \begin{cases} \min\{\ell,k\}\cdot\frac{c(c-1)}{2}+\max\{\ell,k\}\cdot(\ell_{\max}-c+1) & \text{, if } c\leq \ell_{\max} \\ \min\{\ell,k\}\cdot\frac{\ell_{\max}(\ell_{\max}+1)}{2} & \text{, if } c>\ell_{\max} \end{cases}$$

Let us consider the messages at distance $\ell_{\max}$ to their destinations. They are of two types, the one moving according to their negative component and the one moving according to their positive component.

If $c \leq \ell_{\max}$ the first ones move after at most ℓ time steps. If $c < \ell_{\max}$ they move more quickly, indeed they move at least once every $\ell_{\max}k$ steps ($\ell_{\max}k \leq c \cdot k = \ell$). This is due to the fact that when $c < \ell_{\max}$ at a given vertex, at most $\ell_{\max}k$ messages may have to move according to their negative component toward a node at distance $\ell_{\max}$.

About the messages which move according to their positive component, since a node is the destination of at most k messages, they may wait at most k steps.

Consequently, $\ell_{\max}$ decreases by at least one every $\min\{\ell, \ell_{\max}k\}$ steps, which gives the result. □

This gives an algorithm which is fully distributed. Dividing the running time of this algorithm by the combined lower bound we obtain the following ratio :

$$\begin{cases} \frac{\min\{\ell,k\}\cdot\binom{c}{2}+\max\{\ell,k\}\cdot(\ell_{\max}-c+1)}{\ell_{\max}\cdot\max\left(\min\{\ell,k\}\ ,\ \frac{\max\{\ell,k\}}{4\sqrt{c+1}}\right)} & \text{, if } c\leq \ell_{\max} \\ \frac{\min\{\ell,k\}\cdot(\ell_{\max}+1)}{2\cdot\max\left(\min\{\ell,k\}\ ,\ \frac{\max\{\ell,k\}}{4\sqrt{c+1}}\right)} & \text{, if } c>\ell_{\max} \end{cases}$$

We observe that in all cases the running time of the algorithm is at most $\max\{\ell,k\}\cdot\ell_{\max}$. In particular, when $\ell = k$ (that is, $c = 1$) the running time is exactly $\max\{\ell,k\}\cdot\ell_{\max} = \min\{\ell,k\}\cdot\ell_{\max}$, and therefore it is tight (see lower bound of Lemma 4.1).

Corollary 4.1 *There exists a tight algorithm for (k,k)-routing in full-duplex triangular grids.*

The previous algorithms can be generalized for half-duplex triangular grids as well as for full and half-duplex hexagonal grids. The generalization to half-duplex grids is obtained by just adding a factor 2 in both the lower bound and the running time of the algorithm, as we did for the permutation routing algorithm. Thus, let us just focus on the case of full-duplex hexagonal grids, for which we have the following theorems :

Theorem 4.2 *There exists an algorithm for (ℓ,k)-routing in full-duplex hexagonal grids whose running time is at most :*

$$\text{Running time} \leq \begin{cases} 2\min\{\ell,k\}\cdot\frac{c(c-1)}{2}+2\max\{\ell,k\}\cdot(\ell_{\max}-c+1) & \text{, if } c\leq \ell_{\max} \\ 2\min\{\ell,k\}\cdot\frac{\ell_{\max}(\ell_{\max}+1)}{2} & \text{, if } c>\ell_{\max} \end{cases}$$

Lemma 4.4 (First lower bound) *No algorithm based on shortest path routing can route all messages using less than $2\min\{\ell,k\}\cdot\ell_{\max} - \min\{\ell,k\}$ steps in the worst case.*

Definition 4.2 *Given a vertex v, we call the* rectangle *of the hexagonal grid of side (a,b) starting at v to the subset of the hexagonal grid $R^v_{hex\,a,b} = \{v + \alpha\mathbf{i} + \beta\mathbf{j} + \gamma\mathbf{k}, 0 \leq \alpha < a, -\gamma < \beta < b, 0 \leq \gamma < b\} \cap H$ where H is the vertex set of the hexagonal grid. We call such a rectangle a* square *if $a = b$.*

The following lemma gives a second lower bound on the running time of any algorithm using shortest path routing on full-duplex hexagonal grids.

Lemma 4.5 (Second lower bound) *The worst-case running time of any algorithm using shortest path routing on full-duplex hexagonal grids satisfies :*

$$\textit{Running time} \geq \left\lceil \max\{\ell,k\}(2d+\frac{d-2}{2d+1}) \right\rceil \quad ,$$

where $d = \left\lfloor \frac{\sqrt{73c+64\ell_{\max}^2+121+144\ell_{\max}}}{8\sqrt{c+1}} - \frac{3}{8} \right\rfloor$ *and* $c = \left\lceil \frac{\max\{\ell,k\}}{\min\{\ell,k\}} \right\rceil$.

Notice that when $\frac{\ell_{\max}}{c}$ is big, this value tends to $2\max\{\ell,k\}\frac{\ell_{\max}}{\sqrt{c+1}}$, obtaining a performance around twice better than in triangular grids.

Proof: The proof consists in showing that the vertices of $R^v_{hex\,d,d}$ can simultaneously send $\max(\ell,k)$ messages to some vertices of $R^v_{hex\,d+\ell_{\max},d+\ell_{\max}} \setminus R^v_{hex\,d,d}$. This is done as for the triangular grid, using again Theorem 4.1. We do not give all the details, since the idea behind is the same as the proof of Lemma 4.2.

Since the number of vertices inside $R^v_{hex\,d,d}$ is $4d^2+d-2$, and the number of edges outgoing from $R^v_{hex\,d,d}$ is $2d+1$, the congestion on these edges is $\max\{\ell,k\}\frac{4d^2+d-2}{2d+1} = \max\{\ell,k\}(2d+\frac{d-2}{2d+1})$. □

5 Conclusions and Further Research

In this article we have studied the permutation routing, the r-central routing and the general (ℓ,k)-routing problems on plane grids, that is square grids, triangular grids and hexagonal grids. We have assumed the *store-and-forward* Δ-port model, and considered both full and half-duplex networks. The main new results of this article are the following :

1. Tight (also including the constant factor) permutation routing algorithms on full-duplex hexagonal grids, and half duplex triangular and hexagonal grids.
2. Tight (also including the constant factor) r-central routing algorithms on triangular and hexagonal grids.
3. Tight (also including the constant factor) (k,k)-routing algorithms on square, triangular and hexagonal grids.
4. Good approximation algorithms for (ℓ,k)-routing in square, triangular and hexagonal grids, together with new lower bounds on the running time of any algorithm using shortest path routing.

All these algorithms are completely distributed, i.e. can be implemented independently at each node. Finally, we have also formulated the (ℓ,k)-routing problem as a Weighted Edge Coloring problem on bipartite graphs.

There still remain several interesting open problems concerning (ℓ,k)-routing on plane grids. Of course, the most challenging problem seems to find a tight (ℓ,k)-routing algorithm for any plane grid, for $\ell \neq k$. Another interesting avenue for further research is to take into account the queue size. That is, to devise (ℓ,k)-routing algorithms with bounded queue size, or that optimize both the running time and the queue size, under a certain trade-off.

Acknowledgment. We want to thank Frédéric Giroire, Cláudia Linhares-Sales and Bruce Reed for insightful discussions.

References

[1] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM (JACM)*, 44(3) :486–504, 1997.

[2] J. Aspnes, C. Busch, S. Dolev, P. Fatourou, C. Georgiou, and A. Shvartsman. Eight Open Problems in Distributed Computing. *Bulletin of the EATCS no*, 90 :109–126, 2006.

[3] B. Awerbuch and Y. Azar. Local optimization of global objectives : competitive distributeddeadlock resolution and resource allocation. *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 240–249, 1994.

[4] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke. Optimal oblivious routing in polynomial time. *Journal of Computer and System Sciences*, 69(3) :383–394, 2004.

[5] J.-C. Bermond, J. Galtier, R. Klasing, N. Morales, and S. Pérennes. Hardness and approximation of Gathering in static radio networks. *Parallel Processing Letters*, 16(2) :165–183, 2006.

[6] A. Borodin and J. Hopcroft. Routing, merging and sorting on parallel models of computation. *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 338–344, 1982.

[7] C. Busch, M. Magdon-Ismail, and J. Xi. Oblivious routing on geometric networks. *Proceedings of the 17th annual ACM symposium on Parallelism in algorithms and architectures*, pages 316–324, 2005.

[8] C. Busch, M. Magdon-Ismail, and J. Xi. Optimal oblivious path selection on the mesh. *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2005), Denver, Colorado, USA, April*, 2005.

[9] J. Cogolludo and S. Rajasekaran. Permutation Routing on Reconfigurable Meshes. *Algorithmica*, 31 :44–57, 2001.

[10] A. Datta. A Fault-Tolerant Protocol for Energy-Efficient Permutation Routing in Wireless Networks. *IEEE Transactions on Computers*, 54(11) :1409–1421, November 2005.

[11] D. de Werra, M. Demange, B. Escoffier, J. Monnot, and V. T. Paschos. Weighted Coloring on Planar, Bipartite and Split Graphs : Complexity and Improved Approximation. *Lecture Notes in Computer Science*, 3341 :896–907, 2004.

[12] M. Demange, D. de Werra, J. Monnot, and V. T. Paschos. Weighted node coloring : when stable sets are expensive. *WG'02 LNCS*, 2573 :114–125, 2002.

[13] R. Diestel. *Graph Theory.* Springer-Verlag, 2005.

[14] T. Dobravec, B. Robič, and J. Žerovnik. Permutation routing in double-loop networks : design and empirical evaluation. *Journal of Systems Architecture*, 48 :387–402, 2003.

[15] T. Dobravec, J. Žerovnik, and B. Robič. An optimal message routing algorithm for circulant networks. *Journal of Systems Architecture*, 52 :298–306, 2006.

[16] P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. *Discrete Applied Mathematics*, 53 :79–134, 1994.

[17] M. D. Grammatikakis, M. K. D. F. Hsu, and J. Sibeyn. Packet Routing in Fixed-Connection Networks : a Survey. *Journal of Parallel and Distributed Processing*, 54(2) :77–132, 1998.

[18] J. T. Havill. Online Packet Routing on Linear Arrays and Rings. *Lecture Notes in Computer Science*, 2076 :773–784, 2001.

[19] F. Hwang, T. Lin, and R. Jan. A Permutation Routing Algorithm for Double Loop Network. *Parallel Processing Letters*, 7(3) :259–265, 1997.

[20] F. Hwang, Y. Yao, and B. Dasgupta. Some permutation routing algorithms for low-dimensional hypercubes. *Theoretical Computer Science*, 270 :111–124, 2002.

[21] G. E. Jan and M.-B. Lin. Concentration, load balancing, partial permutation routing, and superconcentration on cube-connected cycles parallel computers. *Journal of Parallel and Distributed*

Compututing, 65 :1471–1482, 2005.

[22] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. *Theory of Computing Systems*, 24(1) :223–232, 1991.

[23] D. Karimou and J. F. Myoupo. An Application of an Initialization Protocol to Permutation Routing in a Single-Hop Mobile Ad Hoc Networks. *The Journal of Supercomputing*, 31 :215–226, 2005.

[24] A. Kesselman and K. Kogan. Non-Preemptive Scheduling of Optical Switches. In *IEEE GLOBECOM*, pages 1840–1844, 2004.

[25] S. Klavžar, A. Vesel, and P. Žigert. On resonance graphs of catacondensed hexagonal graphs : structure, coding, and hamiltonian path algorithm. *MATCH Communications in Mathematical and in Computer Chemistry*, 49(49) :99–116, 2003.

[26] E. Kranakis, H. Sing, and J. Urrutia. Compas Routing in Geometric Graphs. In *11th Canadian Conference of Computational Geometry*, pages 51–54, 1999.

[27] M. Kunde, R. Niedermeier, and P. Rossmanith. Faster sorting and routing on grids with diagonals. In *11th Symposium of Theoretical Computer Science*, 775, pages 225–236. Lecture Notes on Computer Science, 1994.

[28] F. Leighton, B. M. Maggs, and A. W. Richa. Fast Algorithms for Finding O(Congestion + Dilation) Packet Routing Schedules. In *28th Annual Hawaii International Conference on System Sciences*, pages 555–563, 1995.

[29] F. T. Leighton, B. M. Maggs, and S. B. Rao. Universal packet routing algorithms. In *FOCS*, 1988. Extended Abstract.

[30] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet Routing and Job-Shop Scheduling in O(congestion + dilation) Steps. *Combinatorica*, 14(2) :167–186, 1994.

[31] T. Leighton. *Introduction to Parallel Algorithms and Architectures : Arrays-Trees-Hypercubes*. Morgan-Kaufman, San Mateo, California, 1992.

[32] T. Leighton, F. Makedon, and I. G. Tollis. A $2n - 2$ Step Algorithm for Routing in an $n \times n$ Array with Constant-Size Queues. *Algorithmica*, 14 :291–304, 1995.

[33] W. Liang and X. Shen. Permutation Routing in All-Optical Product Networks. *IEEE Transactions on Circuits and Systems*, 49(4) :533–538, 2002.

[34] B. Maggs. A Survey of Congestion+ Dilation Results for Packet Scheduling. *40th Annual Conference on Information Sciences and Systems*, 22(24) :1505–1510, 2006.

[35] B. Maggs, F. auf der Heide, B. Vocking, and M. Westermann. Exploiting locality for data management in systems of limited bandwidth. *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 284–293, 1997.

[36] F. Makedon and A. Symvonis. Optimal algorithms for the many-to-one routing problem on 2-dimensional meshes. *Microprocessors and Microsystems*, 17 :361–367, 1993.

[37] F. G. Nocetti, I. Stojmenović, and J. Zhang. Addressing and Routing in Hexagonal Networks with Applications for Tracking Mobile Users and Connection Rerouting in Cellular Networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(9) :963–971, 2002.

[38] R. Ostrovsky and Y. Rabani. Universal O(congestion + dilation + $\log^{1+\varepsilon} N$) Local Control Packet

Switching Algorithms. In *29th Annual ACM Symposium on the Theory of Computing*, pages 644–653, New York, 1997.

[39] A. Pietracaprina and G. Pucci. Optimal Many-to-One Routing on the Mesh with Constant Queues. *Lecture Notes in Computer Science*, 2150 :645–649, 2001.

[40] H. Racke. Minimizing congestion in general networks. *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 43–52, 2002.

[41] S. Rajasekaran and R. Overholt. Constant queue routing on a mesh. *Journal of Parallel and Distributed Computing*, 15 :160–166, 1992.

[42] B. Robič and J. Žerovnik. Minimum 2-terminal routing in 2-jump circulant graphs. *Computers and Artificial Intelligence*, 19(1) :37–46, 2000.

[43] I. Sau and J. Žerovnik. An Optimal Permutation Routing Algorithm for Full-Duplex Hexagonal Mesh Networks. *Preprint series, University of Ljubljana, Institute of Mathematics, Physics and Mechanics*, 44(1017), 2006. ISSN 1318–4865.

[44] I. Sau and J. Žerovnik. Optimal permutation routing on mesh networks. In *Proc. of International Network Optimization Conference*, Belgium, April 2007. 6 pages.

[45] C. Scheideler. *Universal Routing Strategies for Interconnection Networks.* Springer, 1998.

[46] J. Sibeyn. Routing on Triangles, Tori and Honeycombs. *International Journal of Fundations of Computer Science*, 8(3) :269–287, 1997.

[47] J. Sibeyn and M. Kaufman. Deterministic 1-k routing on meshes (with applications to worm-hole routing). In LNCS, editor, *11th Symposium on Theoretical Aspects of Computer Science*, volume 775, pages 237–248, 1994.

[48] J. F. Sibeyn, B. S. Chlebus, and M. Kaufmann. Deterministic Permutation Routing on Meshes. *Journal of Algorithms*, 22(1) :111–141, 1997.

[49] A. Srinivasan and C.-P. Teo. A constant-factor approximation algorithm for packet routing, and balancing local vs. global criteria. In *STOC '97 : Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 636–643, New York, NY, USA, 1997. ACM.

[50] I. Stojmenović. Honeycomb Networks : Topological Properties and Communication Algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 8(10) :1036–1042, 1997.

[51] T. Suel. Routing and Sorting on Meshes with Row and Column Buses. In *Parallel Processing Symposium*, volume Eighth International Volume, pages 411–417, 1994.

[52] R. Tošić, D. Masulović, I. Stojmenović, J. Brunvoll, B. Cyvin, and S. Cyvin. Enumeration of Polyhex Hydrocarbons up to h=17. *Journal of Chemical Information and Computer Sciences*, 35 :181–187, 1995.

[53] R. Trobec. Two-dimensional regular *d*-meshes. *Parallel Computing*, 26 :1945–1953, 2000.

[54] L. Valiant and G. Brebner. Universal schemes for parallel communication. *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 263–277, 1981.

[55] R. Williams. *The Geometrical Foundation of Natural Structure : A Source Book of Design.* Dover Publications, 1979.

A Defining the embeddings

The results already known for the square grid can be used for a triangular (resp. a hexagonal) grid if we have an adapted function mapping the square grid into the triangular (resp. hexagonal) grid. Here we propose both functions, namely *square2triangle* and *square2hexagon*.

The function *square2triangle* is illustrated in Fig. 7. We perform the same routing as in the grid, i.e. we just ignore the extra diagonal.

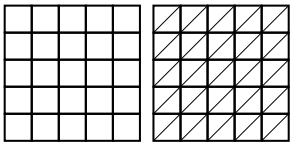

Figure 7 – Square grid mapped into the triangular grid.

In a square grid the distance between two vertices is at most twice the distance in a triangular grid. Similarly the congestion is at most doubled going from the triangular grid to the square grid. Nevertheless the maximal distance is unchanged. Indeed, the NW and SE nodes of Fig. 7 are at the same distance in both grids. Consequently, an algorithm which routes a permutation in $2n-2$ steps is still optimal in the worst case. This is the reason why, instead of considering a square grid with one extra diagonal, we look at a triangle grid in the shape of a triangle, c.f. Fig. 8. Using the routing of the square grid in this triangle grid yields a routing within twice the optimal (i.e. minimum time in the worst case).

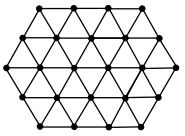

Figure 8 – Triangular grid.

The function *square2hexagon* is a little more complicated. Squares are mapped in two different ways on the hexagonal grid, as shown in Fig. 9. Some are mapped on the left side of a hexagon and some on the right side. Call them respectively *white* and *black* squares. White and black squares alternate on the grid like white and black on a chess board. The missing edge of a white square is mapped to the path of length 3 that goes on the right of the hexagon. The missing edge of a black square is mapped on the path of length 3 that goes on the right of the hexagon. In this way each edge of the square grid is uniquely mapped and each edge of the hexagonal grid is the image of exactly two edges of the square grid.

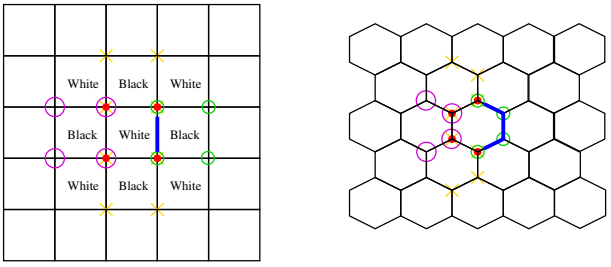

Figure 9 – Square grid mapped into the hexagonal grid.

The distance between 2 vertices in the hexagonal grid is twice the distance in the square grid plus one. Also when we adapt a routing from the square grid to the hexagonal grid using the function *square2hexagon*, the congestion may double since each edge of the hexagonal grid is the image of two edges of the square grid. Consequently, a routing obtained using the function *square2hexagon* will be within a constant multiplicative factor of the optimal.

B An Approach for (ℓ, k)-routing Using Weighted Coloring

In any physical topology, we can represent a given instance of the problem in the following way. Given a network on n nodes, we build a bipartite graph H with a copy of each node at both sides of the bipartition. We add an edge between u and v whenever u wants to send a message to v, and assign to each edge uv a weight $w(uv)$ equal to the length of a shortest path from u to v on the original grid. In this way we obtain an edge-weighted bipartite graph H on $2n$ nodes. Note that the maximum degree of H satisfies $\Delta \leq \max\{\ell, k\}$. An example for $\ell = 2$ and $k = 3$ is depicted in Fig. 10.

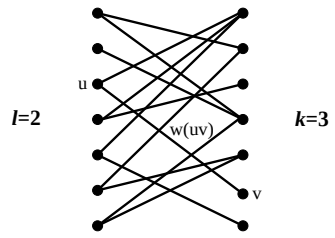

Figure 10 – Bipartite graph modeling a $(2, 3)$-routing instance.

The key idea behind this construction is that each matching in H corresponds to an instance of a permutation routing problem. Hence, it can be solved optimally, as we have proved for all types of grids in Section 2. For each matching M_i, we define its cost as $c(M_i) := \max\{w(e)|e \in M_i)\}$. We assign this cost because on all grids the running time of the permutation routing algorithms we have described are proportional to the length of the longest shortest path (with equality on full-duplex triangular grids).

From the classical Hall's theorem we know that the edges of a bipartite graph can be partitioned into Δ disjoint matchings (that is, a coloring of the edges), Δ being the maximum degree of the graph. In our case we have $\Delta = \max\{\ell, k\}$. Thus, the problem consists in partitioning the edges of H into Δ matchings $M_1, \ldots, M_\Delta$, in such a way that $\sum_{i=1}^{\Delta} c(M_i)$ is minimized. That is, our problem, namely Weighted Bipartite Edge Coloring, can be stated in the following way :

Weighted Bipartite Edge Coloring
Input : An edge-weighted bipartite graph H.
Output : A partition of the edges of H into matchings $M_1, \ldots, M_\Delta$, with $c(M_i) := \max\{w(e)|e \in M_i)\}$.
Objective : $\min \sum_{i=1}^{\Delta} c(M_i)$.

Therefore, $\min \sum_{i=1}^{\Delta} c(M_i)$ is the running time for routing an (ℓ, k)-routing instance using this algorithm.

Unfortunately, in [11] Weighted Edge Coloring is proved to be strongly NP-complete for bipartite graphs, which is the case we are interested in. In fact, the problem remains strongly NP-complete even restricted to cubic and planar bipartite graphs. Concerning approximation results, the authors [11] provide an inapproximability bound of $\frac{7}{6} - \varepsilon$, for any $\varepsilon > 0$. Furthermore, they match this bound with an approximation algorithm within $7/6$ on graphs with maximum degree 3, improving the best known approximation ratio of $5/3$ [12]. In [24] this innaproximability bound is proved independently on general bipartite graphs. Thus, if $\max\{\ell, k\} \leq 3$ we can find a solution of Weighted Bipartite Edge Coloring within $\frac{7}{6}$ times the optimal solution, and this will be also a solution for the (ℓ, k)-routing problem.

Remark B.1 *Although of theoretical value, the main problem of this algorithm is that finding these matchings is a* ***centralized*** *task. In addition, the* true *ratio, i.e. related to the optimum of the (ℓ, k)-routing, should be proved to provide an upper bound for the running time of this algorithm.*

Appendix B

Label Space Minimization in GMPLS Networks

B.1 MPLS Label Stacking on the Path

B.2 Designing Hypergraphs Layouts to GMPLS Routing Strategies

MPLS Label Stacking on the Line Network*

Jean-Claude Bermond[1], David Coudert[1], Joanna Moulierac[1], Stéphane Perennes[1], Hervé Rivano[1], Ignasi Sau[1,2], and Fernando Solano Donado[3]

[1] Joint project MASCOTTE, I3S(CNRS-UNS) INRIA, Sophia-Antipolis, France
[2] Graph Theory and Combinatorics Group at Applied Mathematics IV Department of UPC, Barcelona, Spain
[3] Institute of Telecommunications, Warsaw University of Technology, Poland

Abstract. All-Optical Label Switching (AOLS) is a new technology that performs forwarding without any Optical-Electrical-Optical conversions. The most promising scheme to manage the control plane of these optical networks is Generic MultiProtocol Label Switching (GMPLS). In this paper, we study the problem of routing a set of requests in GMPLS networks with the aim of minimizing the number of labels required to ensure the forwarding. In order to spare the label space, we consider label stacking, allowing the configuration of GMPLS tunnels. We study particularly this network design problem when the network is a line. We provide an exact algorithm for the case in which all the requests have a common source and present some approximation algorithms and heuristics when an arbitrary number of sources are distributed over the line. We analyze by simulations the performance of our proposed algorithms and compare them with previous ones.

Keywords: Label space reduction, label stacking, GMPLS, AOLS.

1 Introduction

All-Optical Label Switching (AOLS) [1] is an approach to transparently route packets all-optically, allowing a speed-up of the forwarding. This very promising technology for the future Internet applications also brings new constraints and, consequently, new problems have to be addressed. Indeed, as the forwarding functions are implemented directly at the optical domain, a specific correlator is needed for each optical label processed in the node. Therefore, it is of major importance to reduce the number of employed correlators in every node, hence reducing the number of labels (as referred in the rest of the paper). The most promising scheme to manage the control plane of these optical networks is Generic MultiProtocol Label Switching (GMPLS). Therefore, for reducing the total number of labels in routers, solutions deployed by GMPLS for reducing the number of labels, such as label merging or label stacking, have to be studied.

* This work has been partly funded by the European project FET AEOLUS, the COLOR INRIA LARECO and the project "Optimization Models for NGI Core Network" (Polish Ministry of Science and Higher Education, grant N517 397334).

L. Fratta et al. (Eds.): NETWORKING 2009, LNCS 5550, pp. 809–820, 2009.

In this paper we consider the problem of routing a set of given requests with the aim of minimizing the total number of labels. We study this problem when the network is a line and when label stacking allowing to configure GMPLS tunnels is considered. Restricting the problem to the case when the network is a line will provide efficient algorithms that are necessary to better apprehend the general problem.

The first studies related to label space reduction in GMPLS networks are based on a technique called label merging (not discussed here). Saito *et al.* were the first considering this problem and they propose in [2] a linear programming mathematical model to find the most efficient routing solution in terms of labels using label merging. It is worth mentioning the heuristic proposed by Bhatnagar *et al.* in [3] with the same aim. The contributions using label merging were further extended in [4].

In [5] the authors deal with the problem of minimizing the number of used labels, when routes are given and the stack depth is limited to two. In [6], the authors extend this problem by assuming that routes should be found as well, considering that links have capacities. In these two contributions, the authors have as objective the minimization of the usage of the label space while keeping the stack depth to a maximum of two, which can be seen as a network design problem since the goal is to find the minimum capacities in the nodes to satisfy a traffic matrix.

This paper is organized as follows. In Section 2, we recall the basic concepts of GMPLS label forwarding mechanism. In Section 3, we formally state the problem addressed in this paper. In Section 4, we present a optimal polynomial-time algorithm when one source is considered in the line. In Section 5, we propose an approximation algorithm and heuristics when multiple sources are considered. Simulation results concerning these algorithms are reported in Section 6. Finally, Section 7 gives conclusion and perspectives of the work.

2 Label Switching Mechanism in GMPLS

In GMPLS, requests are established by the configuration of Label Switched Paths (LSPs). Packets are associated to LSPs by means of a label, or tag, placed in the header of the packet. In this way, routers - called Label Switched Routers (LSR) - can distinguish and forward packets. In addition, in GMPLS, it is allowed to carry a set of labels in packets header, conforming a stack of labels. Even though a packet may contain more than one label, LSRs must only read the first (or top) label in the stack in order to take forwarding decisions. Stacking labels and label processing, in general, is standardized by the following set of operations that an LSR can perform over a given stack of labels:

- SWAP: replace the label at the top by a new one,
- PUSH: replace the label at the top by a new one and then push one or more onto the stack, and
- POP: remove the label at top in the label stack.

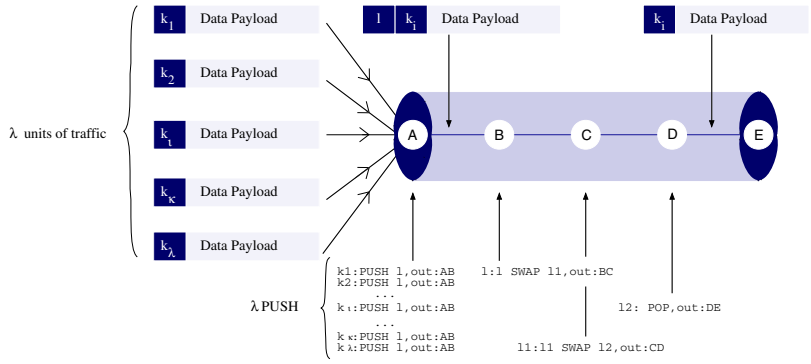

Fig. 1. GMPLS Operations performed at the entrance and at the exit of a tunnel

The labels stored in the forwarding table are significant only locally at the node and swapped all along the LSP.

Label Stacking

When two or more LSPs follow the same set of links, they can be routed together 'inside' a higher-level LSP, henceforth a *tunnel*. In order to setup a tunnel, multiple labels are placed in the packet's header: a method known in the literature as *label stacking*.

As mentioned before, the LSRs in the core of the network route data solely on the basis of the topmost label in the stack. This helps to reduce both the number of labels that need to be maintained on the core LSRs and the complexity of managing data forwarding across the backbone.

Figure 1 represents the general operations needed to configure a tunnel with the use of label stacking. At the entrance of the tunnel, λ PUSH are performed in order to route the λ units of traffic through the tunnel. Then, only one type of operation (either a SWAP or a POP at the end of the tunnel) is performed in all the nodes along the tunnel, regardless of λ. In this figure, a stack of size 2 is used to route the λ LSPs in one tunnel from node A to node E. The top label l is swapped and replaced at each hop: by l_1 at node B, by l_2 at node C, and is finally popped at node D. The λ units of traffic, at the exit of the tunnel at node E can end or follow different paths according to their bottom label k_i, for all $i \in \{1, 2, ..., w\}$ in the stack.

Therefore, the total cost $c(T)$ of this tunnel $T = (A, E)$ in terms of number of labels is: $c(T) = \lambda + l(T) - 1$, where λ is the number of units of traffic forwarded through this tunnel and $l(T)$ is its length in terms of number of hops (which is 4 on this example).

The traffic can enter in any node of a tunnel but can exit in only one point, the last node of the tunnel. In other words, when some traffic is carried by a tunnel, it follows the tunnel until the end.

This cost function $c(T)$ still holds for some degenerated cases. For example, in the case of an arc (*i.e.*, a path of length 1, $l(T) = 1$), or when one unit of traffic

is routed in a path (*i.e.*, a single LSP with $\lambda = 1$ whose cost is only its length). In the following, we consider as a tunnel, without loss of generality: (1) an arc routing several units of traffic, (2) a path routing a only one unit of traffic, and (3) a path routing several units of traffic (*i.e.*, $\lambda > 1$ and $l(t) > 1$). Note that strictly speaking, only the third case is considered as a tunnel.

In this paper, we fix the maximum stack size to 2. Increasing the stack size, increases also the total bandwidth consumption in the network. When the size of the stack is not limited, *label stripping* [7,8] encoding the whole path in the stack provides a feasible solution.

3 Modelling the LSPR Problem

This section describes the problem of routing a set of requests in GMPLS network with the aim of minimizing the number of labels. The problem is formally defined as follows:

Label Space Reduction in a GMPLS Network: LSPR

INPUT: a network (digraph) $G = (V, E)$ and a set of requests $\mathcal{R}$, where in the request $r \in \mathcal{R}$, $r = (s_i, u_j)$, $s_i \in V$ sends w_r units of traffic to $u_j \in V$.

OUTPUT: A set $\mathcal{T}$ of tunnels enabling to route the traffic and a dipath composed of tunnels in $\mathcal{T}$ for each request (s_i, u_j).

OBJECTIVE: minimize the total cost of $\mathcal{T}$, that is $c(\mathcal{T}) = \sum_{T_k \in \mathcal{T}} c(T_k)$ where the cost $c(T_k)$ of a tunnel T_k which contains λ_k units of traffic and is of length $l(T_k)$ (number of arcs in G associated to the path joining the end-vertices of T_k) is $c(T_k) = \lambda_k + l(T_k) - 1$.

Computation of a solution to the example of Figure 2. Consider the line network with one source s, w_1 units of traffic destined to u_1 at distance l_1 from s ($l_1 - 1$ nodes between s and u_1) and w_2 units of traffic destined to u_2 at distance $l_1 + l_2$ from s. See Figure 2 for an illustration. The optimal solution depends on the values l_i and w_i. Indeed, two solutions have to be examined.

In the first solution, a specific tunnel (s, u_i) is configured for each destination u_i, giving two tunnels (s, u_1) and (s, u_2) with a total cost: $(w_1 + l_1 - 1) + (w_2 + l_1 + l_2 - 1) = w_1 + w_2 + 2l_1 + l_2 - 2$.

The second solution is composed of the two tunnels (s, u_1) and (u_1, u_2). The requests destined to u_2 will first use the tunnel (s, u_1) and then the tunnel (u_1, u_2). The traffic carried by (s, u_1) is $\lambda_1 = w_1 + w_2$ and the traffic carried by (u_1, u_2) is $\lambda_2 = w_2$. Therefore, the total cost is $(w_1 + w_2 + l_1 - 1) + (w_2 + l_2 - 1) = w_1 + 2w_2 + l_1 + l_2 - 2$.

The optimal solution is either the first one if $l_1 \leq w_2$ or the second one if $l_1 \geq w_2$.

Lemma 1. *In any network $G = (V, E)$, there exists an optimal solution $\mathcal{T}$ for the problem* LSPR *such that all the units of traffic of the request (s_i, u_j) are routed in $\mathcal{T}$ via a unique dipath (set of consecutive tunnels) from s_i to u_j.*

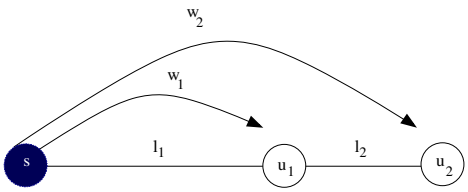

Fig. 2. Depending on the values l_1 and w_2, the optimal solution may be composed either of tunnels (s, u_1) and (s, u_2), or of tunnels (s, u_1) and (u_1, u_2)

Proof. Let $\mathcal{T}$ be an optimal solution and suppose that the requests arriving at $u_j$are routed via $p > 1$ different paths $P_1, \ldots, P_m$. Let λ_m, $1 \leq m \leq p$, be the number of traffic units forwarded by P_m. Let h_m (h like hops), $1 \leq m \leq p$, be the number of consecutive tunnels in the path P_m. Let the order of the paths be such that P_1 is a path with the minimum number of consecutive tunnels h_1.

Then, for any other path P_m ($m > 1$) reroute the λ_m requests routed via P_m via P_1. We obtain a new feasible solution $\mathcal{T}'$ whose cost is

$$c(\mathcal{T}') \leq c(\mathcal{T}) + \lambda_m h_1 - \lambda_m h_m.$$

Indeed, the cost of each tunnel used in P_m is decreased by λ_m, plus possibly, if some tunnel T of P_m becomes empty, by $l(T) - 1 \geq 0$. On the other hand, the cost of each tunnel of P_1 is increased only by λ_m as the tunnel already exists. Therefore, as $h_1 \leq h_m$, we get $c(\mathcal{T}') \leq c(\mathcal{T})$ with strict inequality if $h_1 < h_m$ (the path P_m is strictly longer than P_1) or if, in the rerouting, some tunnels of length more than 1 become empty. So $\mathcal{T}'$ is also an optimal solution.

Repeating the operation for each P_m we obtain an optimal solution $\mathcal{T}^*$, where all the requests arriving at a node u_i are routed in $\mathcal{T}$ via a unique dipath. □

The cost of an optimal solution $\mathcal{T}$ for problem LSPR with $|\mathcal{T}|$ tunnels and $|\mathcal{R}|$ requests is:

$$c(\mathcal{T}) = \sum_{k=1}^{|\mathcal{T}|} (l(T_k) - 1) + \sum_{r=1}^{|\mathcal{R}|} h_r w_r.$$

where h_r is the number of consecutive tunnels for the request r in $\mathcal{T}$, w_r is the number of units of traffic of the request r and $l(T_k)$ is the length of the tunnel T_k in terms of number of hops. The cost $c(\mathcal{T})$ is the sum of the cost for the configuration of the tunnels $(\sum_{k=1}^{|\mathcal{T}|}(l(T_k) - 1))$ and the cost for the requests to enter the tunnels $(\sum_{r=1}^{|\mathcal{R}|} h_r w_r)$.

4 LSPR-L1 Problem: The Line Network, One Source

In this section, we focus on the specific case when the network $G = (V, E)$ is a directed line (a dipath) and when the number of sources is equal to 1. Focusing on the same problem with simplest constraints will provide algorithms that will be

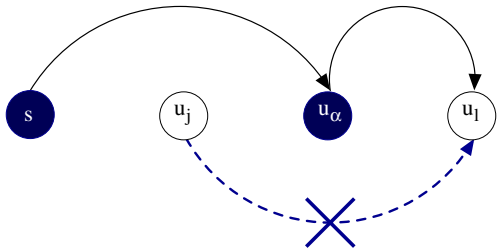

Fig. 3. The tunnel in dotted points is not present in an optimal solution

useful to find efficient solutions for the general problem. Let us denote by $P_{s\to u_n}$ the line where s is the source and where there are n requests (s,u_i) with the u_i indexed in the increasing order of their distance from s. This problem is referred as LSPR-L1 in the sequel (standing for Label Space reduction in a GMPLS Line Network with 1 source). The main result of this section is an algorithm based on dynamic programming techniques that finds an optimal solution in time $\mathcal{O}(n^3)$, as stated in Proposition 1. First, we need two technical lemmas.

Lemma 2. *When the network is a directed line, with source s, an optimal solution $\mathcal{T}$ for* LSPR-L1 *problem is such that, if (s,u_α) is the longest tunnel from s, then there is no tunnel (u_j,u_l) in $\mathcal{T}$ with $j<\alpha<l$.*

Proof. Suppose there exists such a tunnel (u_j,u_l) (see Figure 3). As α is the maximum index, then $u_j\neq s$, otherwise (s,u_l) would have been longer than (s,u_α). Therefore, the number of consecutive tunnels towards u_l, $h_{r=(s,u_l)}$ denoted simply h_l, satisfies: $h_l\geq 2$. Consider the solution $\mathcal{T}'$ obtained from $\mathcal{T}$ by deleting the tunnel (u_j,u_l) and adding, if it does not exist, the tunnel (u_α,u_l). The request (s,u_l) is then routed through two consecutive tunnels (s,u_α) and (u_α,u_l). It is an admissible solution whose cost satisfies:

$$c(\mathcal{T}')\leq c(\mathcal{T})-\lambda_l h_l-(l(u_j,u_l)-1)+2\lambda_l+l(u_\alpha,u_l)-1,$$

where λ_l is the number of requests arriving at u_l. As $h_l\geq 2$ and $l(u_\alpha,u_l)<l(u_j,u_l)$, $c(\mathcal{T}')<c(\mathcal{T})$. □

Lemma 3. *For a line $P_{s\to u_n}$ with w_i units of traffic for the request (s,u_i), the cost of an optimal solution is:*

$$c^*(P_{s\to u_n})=\min_{\alpha}\ [\ \sum_{i=\alpha}^{n} w_i+l(s,u_\alpha)-1+c^*(P_{s\to u_{\alpha-1}})+c^*(P_{u_\alpha\to u_n})\ ],$$

where $u_\alpha\in P_{u_1\to u_n}$ is a splitting point that decomposes the problem into two sub-problems.

Proof. By Lemma 2 an optimal solution contains a tunnel (s,u_α) of cost $(\sum_{i=\alpha}^{n} w_i+l(s,u_\alpha)-1)$ plus an optimal solution on the sub-line $P_{s\to u_{\alpha-1}}$ and an optimal solution on the sub-line $P_{u_\alpha\to u_n}$. □

Algorithm 1. Polynomial-time algorithm computing an optimal solution for the LSPR-L1 problem

Input: Line $P_{s\to u_n}$ from s to u_n, where s is the source (referred also as u_0) and (s, u_i) are the set of requests ($i \geq 1$), each of them having w_i units of traffic

Output: Set of tunnels enabling the routing from s of all the requests (s, u_i)

begin

C is a table of size n^2 indicating the costs all the sub-solutions;
S is a table of size n^2 indicating the splitting points u_α associated to the optimal sub-solutions;
W is a table of size n storing partial sums of weigths, $W[0] = 0$,
$W[j] = \sum_{i=1}^{j} w_i = W[j-1] + w_j$, and so $\sum_{i=\alpha}^{\beta} w_i = W[\beta] - W[\alpha - 1]$;
for $i \in [0, n]$ **do**
  $C[u_i, u_i] = 0$;
  $C[u_i, u_{i+1}] = w_{i+1} + l(u_i, u_{i+1}) - 1$;
  $S[u_i, u_{i+1}] = u_{i+1}$;
for $i \in [2, n]$ **do**
  for $\forall k \in [0, n-i]$ **do**
    $min = +\infty$;
    for $\forall \alpha \in [k+1, k+i]$ **do**
      $value =$
      $(W[k+i] - W[\alpha - 1]) + l(u_k, u_\alpha) - 1 + C[u_k, u_{\alpha-1}] + C[u_\alpha, u_{k+i}]$;
      if $value < min$ **then**
        $min = value$;
        $C[u_k, u_{k+i}] = c(P_{u_k \to u_{k+i}}) = value$;
        $S[u_k, u_{k+i}] = u_\alpha$;
Compute the optimal set of tunnels from the table S;

end

Proposition 1. *When the network is a directed line $P_{s\to u_n}$, and all requests are issued from s, then an optimal solution of the* LSPR-L1 *problem can be computed in time $\mathcal{O}(n^3)$ by Algorithm 1.*

Proof. According to Lemma 3, to compute an optimal solution for $P_{s\to u_n}$, we need first to compute optimal sub-solutions for $P_{s\to u_{\alpha-1}}$ and for $P_{u_\alpha \to u_n}$, $u_\alpha \in \{u_1, \ldots, u_n\}$, and recursively. The algorithm computes first solutions for $P_{u_i \to u_{i+1}}$, and for computing solutions for $P_{u_i \to u_{i+2}}$, the already computed values for sub-lines $P_{u_i \to u_{i+1}}$ (say $C[u_i, u_{i+1}]$) and $P_{u_{i+1} \to u_{i+2}}$ (say $C[u_{i+1}, u_{i+2}]$) are used without any recomputation.

For example, to compute the solution on $P_{s\to u_2}$, we need the values $C[s, u_1]$ and $C[u_1, u_2]$ since we have $C[s, u_2] = \min\{(w_1 + w_2 + l(s, u_1) - 1 + C[u_1, u_2]), (w_2 + l(s, u_2) - 1 + C[s, u_1])\}$. Now, if we want to compute the solution on $P_{s\to u_3}$, we need to compute first $C[u_1, u_3]$ and $C[u_2, u_3]$, but not $C[s, u_1]$ and $C[s, u_2]$ that are already known from previous computations and stored in table C.

Finally, we can compute the optimal solution using dynamic programming (Algorithm 1), with time complexity $\mathcal{O}(n^3)$ and space complexity $\mathcal{O}(n^2)$. □

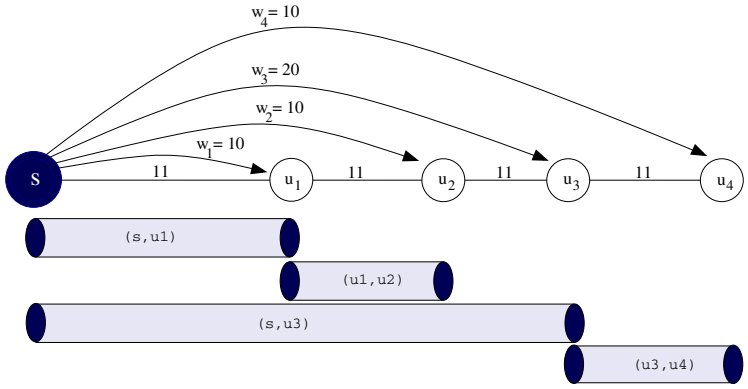

Fig. 4. An example with its optimal solution

The optimal algorithm in the example of Figure 4. Let us compute an optimal solution to the example in Figure 4 using Algorithm 1. We first have to compute the table C containing the costs of the sub-optimal solutions for each sub-line.

First, the sub-paths of length 1, $P_{s\to u_1}$, $P_{u_1\to u_2}$, $P_{u_2\to u_3}$, and $P_{u_3\to u_4}$ are straightforward computed in $C[u_0,u_1]$, $C[u_1,u_2]$, $C[u_2,u_3]$, and $C[u_3,u_4]$.

Then, for the sub-paths of length 2, $P_{s\to u_2}$, $P_{u_1\to u_3}$, and $P_{u_2\to u_4}$, two splitting points are considered by the algorithm. For example, for $P_{s\to u_2}$, the optimal solution implies a splitting point u_1 with cost $w_1+w_2+l(s,u_1)-1+C[u_0,u_0]+C[u_1,u_2]=50$ (the splitting point u_2 implying a greater cost $w_2+l(s,u_2)-1+C[u_0,u_1]+C[u_2,u_2]=51$). The already computed costs $C[u_0,u_0]$, $C[u_1,u_2]$, and $C[u_2,u_2]$ have been used by the algorithm and are not computed again.

For the computation of the optimal solution on the whole line $P_{s\to u_4}$, four splitting points, u_1, u_2, u_3, and u_4 should be considered.

When the table C showing the optimal costs for all the subpaths has been computed as presented in Table 1, the set of tunnels composing the optimal solution can be deduced from the splitting points. The optimal solution for line $P_{s\to u_4}$ has cost 132 and a splitting point u_3. Thus, the optimal solution is composed of a tunnel (s,u_3) and of optimal solutions for the sub-paths $P_{s\to u_2}$ and $P_{u_3\to u_4}$. The first sub-solution has a splitting point u_1 which gives tunnels (s,u_1), (u_1,u_2). The optimal solution for the sub-path $P_{u_3\to u_4}$ is obviously the tunnel (u_3,u_4).

Finally, the optimal solution is composed of tunnels (s,u_1), (u_1,u_2), (s,u_3), and (u_3,u_4).

In the special case where the requests are uniform, we are able to give a closed formula of the cost of an optimal solution, as stated as follows.

Proposition 2. *For a line network $P_{s\to u_n}$, with $n=2^q-1+r$, where $0\le r\le 2^q-1$, and an uniform distribution: $\forall i, w_i=1$, the cost of an optimal solution is $2^q(q-1)+1+(q+1)r$.*

The proof of this proposition is technical and is not given in this paper due to lack of space. In this specific case we can prove that $c^*(P_{s\to u_n})=c^*(P_{s\to u_{n-1}})+\lfloor\log(n)\rfloor+1$.

Table 1. Computation of the table C and S for the optimal solution of the example on Figure 4, the nodes in brackets representing the splitting points of table S

	$s = u_0$	u_1	u_2	u_3	u_4
$s = u_0$	0	20	50 (u_1)	101 (u_2)	132 (u_3)
u_1	-	0	20	61 (u_3)	91 (u_3)
u_2	-	-	0	30	60 (u_3)
u_3	-	-	-	0	20
u_4	-	-	-	-	0

5 LSPR-LM Problem: The Line Network, Multiple Sources

In this section, we study the problem of routing a set of requests on the line network when multiple sources are distributed along the line. Since sources may inject traffic anywhere in the network, Lemma 2 is not valid anymore, hence the problem seems to be inherently more complicated. As the problem cannot be decomposed as easily as previously, we present in this section a $\log(n)-$approximation algorithm and an heuristic that will be compared to the optimal solution and to previous known heuristics in Section 6. The problem is referred in the following as LSPR-LM (standing for Label Space reduction in a GMPLS Line Network with Multiple sources).

5.1 log(n)-Approximation Algorithm for LSPR-LM

Consider the nodes $\{u_0, u_1, \ldots, u_n\}$, that can be source or destination or both, sorted according to their position on the line from the left to the right (u_{i+1} after u_i on the line, u_{i+1} being at distance l_{i+1} from u_i). Suppose that the line is of length L, meaning that $L = \sum_{i=1}^{n} l_i$.

The algorithm consists of configuring all the consecutive tunnels $\{(u_0, u_1)$, (u_1, u_2), $\ldots, (u_{n-1}, u_n)\}$, $\{(u_0, u_2)$,$(u_2, u_4), \ldots,(u_{n-2}, u_n)\}$, $\{(u_0, u_4)$, (u_4, u_8), $\ldots, (u_{n-4}, u_n)\}$, and more generally, those of length a power of 2. See Figure 5 for an illustration of the configuration of the tunnels. Consequently, there exists a path of at most $\log(n)$ tunnels from any source to any destination, ensuring a valid routing for all the requests. When the solution has been computed, then some tunnels that are not used by any destination may be removed.

Theorem 1. *For a problem with n sources and/or destinations, there exists a* $\log(n)$*-approximation algorithm for the* LSPR-LM *problem.*

Proof. The cost of a solution computed by the algorithm is (1) the cost of the configuration of the tunnels plus (2) the cost for entering the tunnels.

To configure each level of consecutive tunnels, at most $L-1$ labels are needed. There are at most $\log(n)$ different levels of tunnels. So, the overall number of labels needed for the configuration of tunnels is at most $(1) \leq (L-1)\log(n)$.

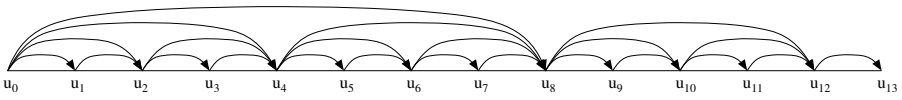

Fig. 5. Computing this set of tunnels gives a $\log(n)-$approximation algorithm for LSPR-LM problem

When that set of tunnels has been configured, any source can join any destination in at most $\log(n)$ hops. Therefore, the total cost needed to enter the tunnels is at most $(2) \leq \sum_{r=1}^{|\mathcal{R}|} w_r \log(n)$.

Then, the cost of this solution is at most: $(1) + (2) \leq \sum_{r=1}^{|\mathcal{R}|} w_r \log(n) + (L-1)\log(n) = \log(n)(\sum_{r=1}^{|\mathcal{R}|} w_r + L - 1)$.

In the best case, an optimal solution will be of cost $\sum_{r=1}^{|\mathcal{R}|} w_r + (L-1)$, giving a $\log(n)-$approximation. □

5.2 Proposed Heuristic: Extended Dynamic Programming

This subsection presents a simple heuristic to find a solution of the problem on the line with multiple sources. Suppose that, when constructing the solution, there is a set of tunnels leading from a source u_0 to a destination u_i. Then, if another source, say u_x with $x > 1$, has to transmit traffic to u_i, then u_x may insert traffic directly in the tunnels going to u_i without additional cost.

Therefore, the heuristic consists of considering only the source u_0, then, to affect the whole set of requests to u_0 and to use the polynomial algorithm just described previously for only one source. In the solution, there would be tunnels from u_0 to all the destinations, and the other sources will insert their traffic in the tunnels passing through them.

6 Simulations

In this section we analyze the performance of the proposed heuristics using simulations. The analysis consists of the comparison of the total number of labels used by the heuristics.

In our simulations, we use a line network consisting of 500 nodes. Each experiment consists of a different number of sources and destinations. The number of sources equals the number of destinations in each experiment. Between a pair of source and destination nodes, a demand is generated (with a probability of 80%) with a random capacity between one and 500 (uniform).

Figure 6 (top) shows the behavior the heuristics proposed in this article together with the Longest Segment First (LSF) heuristic [9]. The number of sources (or destinations) varies from 5 to 113 with increments of three nodes in each experiment. Each experiment was run 100 times. The results show that, even though the $\log(n)$-approximation runs in $\mathcal{O}(p \log n)$ and guarantees a

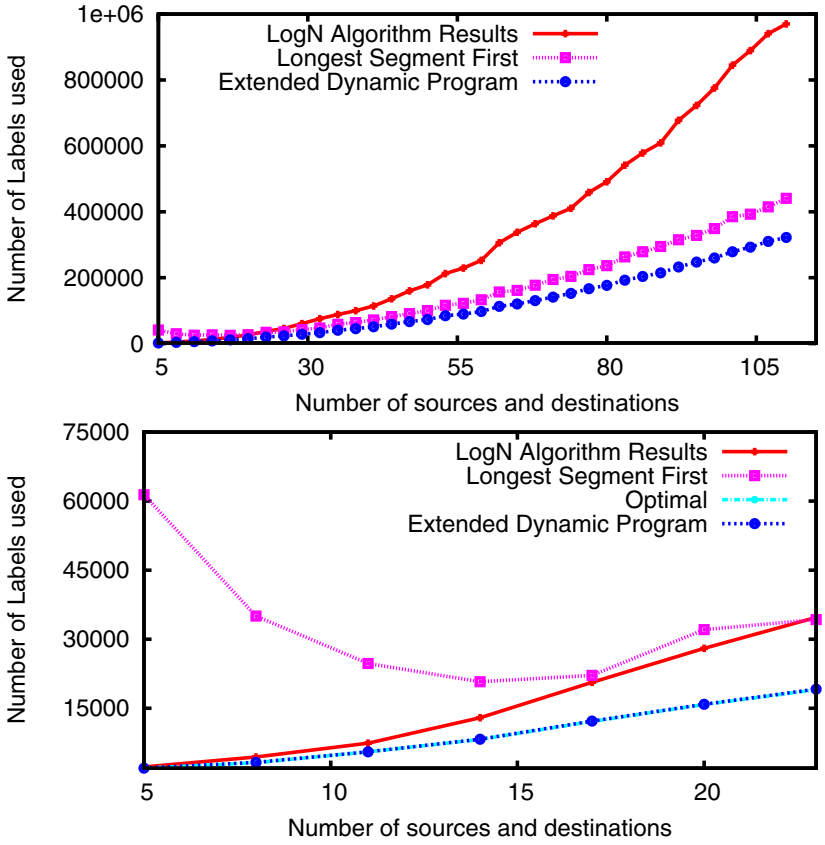

Fig. 6. Comparison on the number of labels used by different heuristics and magnification of the first 20 experiments including the optimal solution

bound in terms of sub-optimality, in practice the results are not as good as the proposed Extended Dynamic Programming heuristic or the LSF heuristic running in $\mathcal{O}(n^3)$ and $\mathcal{O}(np^2)$, respectively. We also observed that the requirements in memory for LSF are lower than those of the Extended Dynamic Programming heuristic; however the quality of the solution of the later always outperforms the former's. Some other previously proposed heuristics (see [6] and [10]) were tested as well with worse results, hence not considered in this analysis.

At the bottom of Figure 6, a magnification of the results in the first 20 experiments is shown. The plot showing the optimal value is also added. In these experiments, the numerical solution computed by the heuristic based in dynamic programming is within 1% (in most of the case) of the optimal value. We conjecture that this is because the demands share the same set of destinations. The proposed heuristics in this paper show a better convergence than that of LSF when the number of sources is low.

7 Conclusion and Perspectives

We presented in this paper the problem of routing a set of requests with the aim of reducing the total number of labels in the network. For the line, we exhibit a polynomial-time algorithm when there is a single source and a $\log(n)$-approximation algorithm, and one heuristic for multiple sources. We show the good performance of these algorithms through simulations. In future work, we plan to extend these proposed algorithms to general networks and to study the computational complexity of the LSPR problem.

References

1. Ramos, F., et al.: IST-LASAGNE: Towards all-optical label swapping employing optical logic gates and optical flip-flops. IEEE J. Sel. Areas Commun. 23(10), 2993–3011 (2005)
2. Saito, H., Miyao, Y., Yoshida, M.: Traffic engineering using multiple MultiPoint-to-Point LSPs. In: IEEE Conference on Computer Communications (Infocom 2000), pp. 894–901 (2000)
3. Bhatnagar, S., Ganguly, S., Nath, B.: Creating Multipoint-to-Point LSPs for traffic engineering. IEEE Commun. Mag. 43(1), 95–100 (2005)
4. Solano, F., Fabregat, R., Marzo, J.: On optimal computation of MPLS label binding for MultiPoint-to-Point connections. IEEE Trans. Commun. 56(7), 1056–1059 (2007)
5. Solano, F., Stidsen, T., Fabregat, R., Marzo, J.: Label space reduction in MPLS networks: How much can a single label do? IEEE/ACM Trans. Netw. (December 2008)
6. Solano, F., Caenegem, R.V., Colle, D., Marzo, J.L., Pickavet, M., Fabregat, R., Demeester, P.: All-optical label stacking: Easing the trade-offs between routing and architecture cost in all-optical packet switching. In: IEEE Conference on Computer Communications (Infocom 2008), Phoenix, AZ, USA, April 2008, pp. 655–663 (2008)
7. Van Caenegem, R., et al.: Benefits of label stripping compared to label swapping from the point of node dimensioning. Photonic Network Communications Journal 12(3), 227–244 (2006)
8. Van Caenegem, R., et al.: From IP over WDM to all-optical packet switching: Economical overview. J. Lightw. Technol. 24(4), 1638–1645 (2006)
9. Solano, F., Fabregat, R., Donoso, Y., Marzo, J.: Asymmetric tunnels in P2MP LSPs as a label space reduction method. In: Proc. IEEE International Conference on Communications (ICC 2005), May 2005, pp. 43–47 (2005)
10. Solano, F., Fabregat, R., Marzo, J.: A fast algorithm based on the MPLS label stack for the label space reduction problem. In: Proc. IEEE IP Operations and Management, IPOM 2005 (October 2005)

Designing Hypergraph Layouts to GMPLS Routing Strategies *

Jean-Claude Bermond[1], David Coudert[1], Joanna Moulierac[1], Stéphane Pérennes[1], Ignasi Sau[1,2], and Fernando Solano Donado[3]

[1] Mascotte joint project , I3S(CNRS-UNS) INRIA, Sophia-Antipolis, France
[2] Applied Mathematics IV Department of UPC, Barcelona, Spain
[3] Institute of Telecommunications, Warsaw University of Technology, Poland

Abstract. All-Optical Label Switching (AOLS) is a new technology that performs packet forwarding without any Optical-Electrical-Optical (OEO) conversions. In this paper, we study the problem of routing a set of requests in AOLS networks using GMPLS technology, with the aim of minimizing the number of labels required to ensure the forwarding. We first formalize the problem by associating to each routing strategy a logical hypergraph whose hyperarcs are dipaths of the physical graph, called *tunnels* in GMPLS terminology. Such a hypergraph is called a *hypergraph layout*, to which we assign a cost function given by its physical length plus the total number of hops traveled by the traffic. Minimizing the cost of the design of an AOLS network can then be expressed as finding a minimum cost hypergraph layout.
We prove hardness results for the problem, namely for general directed networks we prove that it is NP-hard to find a $C \log n$-approximation, where C is a a positive constant and n is the number of nodes of the network. For symmetric directed networks, we prove that the problem is APX-hard. These hardness results hold even is the traffic instance is a partial broadcast. On the other hand, we provide an $\mathcal{O}(\log n)$-approximation algorithm to the problem for a general symmetric network. Finally, we focus on the case where the physical network is a path, providing a polynomial-time dynamic programming algorithm for a bounded number of sources, thus extending the algorithm given in [1] for a single source.

1 Introduction

All-Optical Label Switching (AOLS) [9] is an approach to route packets transparently and all-optically, thus allowing a speed-up of the forwarding. This very promising technology for the future Internet applications also brings new constraints and new problems. Indeed, since the forwarding functions are implemented directly at the optical domain, a specific correlator (device) is needed

* This work has been partly funded by the European project IST FET AEOLUS and the project "Optimization Models for NGI Core Network" (Polish Ministry of Science and Higher Education, grant N517 397334).

for each optical label processed in the node. Therefore, it is of major importance to reduce the number of employed correlators in every node, implying a reduction in the number of labels (as referred in the rest of the paper) that are going to be used by the traffic. Due to its flexibility as a control plane and to the fact that it handles traffic forwarding, the Generic MultiProtocol Label Switching (GMPLS) is the most promising protocol to be applied in AOLS-driven networks.

In GMPLS, traffic is forwarded through logical connections called Label Switched Paths (LSPs). When GMPLS is used with packet-based network, packets are associated to LSPs by means of a label, or tag, placed on top of the header of the packet. In this way, routers - called Label Switched Routers (LSRs) - can distinguish and forward packets.

The GMPLS standards allow packets to carry a set of labels in their header, conforming a stack of labels. Even though a packet may contain more than one label, LSRs must only read the first (or top) label in the stack in order to take forwarding decisions. This helps to reduce both the number of labels that need to be maintained on the core LSRs and the complexity of managing data forwarding across the backbone.

Stacking labels and label processing, in general, are standardized by the following set of operations that an LSR can perform over a given stack of labels:

- SWAP: replace the label at the top by a new one,
- PUSH: replace the label at the top by a new one and then push one or more onto the stack, and
- POP: remove the label at top in the label stack.

The labels stored in the forwarding table are significant only locally at the node and swapped all along the LSP (see Fig. 1).

Solutions deployed by GMPLS for reducing the number of labels are *label merging* [4, 11, 13] (not discussed here) and *label stacking* [12, 15]. With label stacking, when two or more LSPs follow the same set of links, they can be routed together "inside" a higher-level LSP, henceforth a *tunnel*. In order to setup a tunnel, multiple labels are placed in the packet's header.

Fig. 1 represents the general operations needed to configure a tunnel with the use of label stacking. At the entrance of the tunnel, λ PUSH are performed in order to route the λ units of traffic through the tunnel. Then, only one operation (either a SWAP or a POP at the end of the tunnel) is performed in all the nodes along the tunnel, regardless of λ. In this figure, a stack of size 2 is used to route the λ LSPs in one tunnel from node A to node E. The top label l is swapped and replaced at each hop: by l_1 at node B, by l_2 at node C, and is finally popped at node D. The λ units of traffic, at the exit of the tunnel at node E can end or follow different paths according to their bottom label k_i, for all $i \in \{1, 2, ..., w\}$ in the stack.

A consequence of the way in which the GMPLS operations can be configured at LSRs is the following: traffic can enter in any node of a tunnel but can exit in only one point, the last node of the tunnel. In other words, when some traffic is carried by a tunnel, it follows the tunnel until its end.

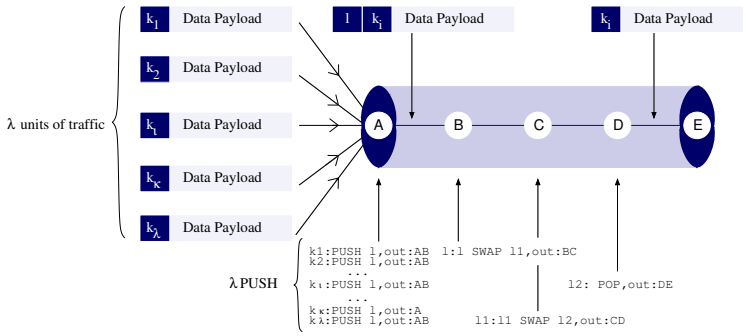

Fig. 1. GMPLS operations performed at the entrance and at the exit of a tunnel.

Since the number of labels used for GMPLS forwarding affects the cost of the AOLS architecture, in this paper we mainly focus on the minimization of the number of labels used. In our previous example, the total cost $c(T)$ of this tunnel T from node A to node E in terms of number of labels is $c(T) = \lambda + \ell(T) - 1$, where λ is the number of units of traffic forwarded through this tunnel and $\ell(T)$ is its length in terms of number of hops (which is 4 on this example). We will formally define the cost function of the problem in Section 2.

Previous work and our contribution. The label minimization problem in GMPLS networks has been widely studied in the literature during the last few years [4, 11–15]. All these articles focus mainly on proposing and analyzing heuristics to the problem, but there is a lack of theoretical results, like computational complexity or bounds on the approximation ratio of the proposed algorithms. For instance, in [14] the authors propose heuristics for routing a set of demands in AOLS networks when routers have limited number of available optical correlators. Very recently [1], the problem has been studied for the directed path from a more theoretical point of view. Namely, in [1] the authors present a polynomial-time optimal algorithm for the case when all traffic is issued from a single source and an $\mathcal{O}(\log n)$-approximation algorithm with arbitrary number of sources, where n is the number of nodes of the network.

In this article we provide the first theoretical framework for the label minimization problem in general GMPLS networks. We translate the problem into finding a set of dipaths in a directed hypergraph. With this new formulation, it turns out that the problem is very similar to classical Virtual Path Layout (VPL) problems originating from ATM networks. We provide hardness results and approximation algorithms for the problem in general graphs. The approximation algorithms strongly rely on the already known algorithms for VPL problems. Finally, we focus on the path topology, extending the dynamic programming approach presented in [1] to any bounded number of sources. If there are k sources, the main result is an optimal algorithm with running time $n^{\mathcal{O}(k)}$. That is, the problem is polynomial in the path for any fixed number of sources.

Organization of the paper. In Section 2 we formally state the problem in terms of hypergraph layout and fix the notation to be used throughout the article. In Section 3 we prove that for general directed networks it is NP-hard to find a $C \log n$-approximation, where C is a a positive constant and n is the number of nodes of the network. For symmetric directed networks, we prove that the problem is APX-hard, and therefore it does not accept a PTAS unless P=NP. In Section 4 we provide an approximation algorithm to the problem for symmetric directed graphs with an approximation ratio $\mathcal{O}(\log n)$, where n is the number of nodes of the network. In Section 5 we focus on the directed path topology and present a dynamic programming approach solving the problem in polynomial time when the number of sources is fixed. Finally, Section 6 is devoted to conclusions and further research.

2 GMPLS Logical Network Design as a Hypergraph Layout Problem

The logical network design problem that we address can be roughly described as follows: we are given a digraph (directed graph) G together with a set of traffic demands (or requests) between couples of vertices in G, and we must find a set of tunnels of minimum cost allowing to route all traffic requests. Note that usually communication networks are symmetric digraphs (i.e. when operators set a link on one direction, they also set the opposite link). So it is interesting to study the symmetric case, which turns out to be computationally easier than the general directed case. Let us now precise each one of the above terms.

A *tunnel* is simply a directed path (or dipath) in G, and due to the technological constraints discussed in Section 1, traffic can enter anywhere in the tunnel but must leave only at the end of the tunnel. To define the problem formally we need the following notation:

- $G = (V, E)$ is the underlying digraph (which can be symmetric or not).
- $|V| = n$, and vertices are numbered $1, \ldots, n$.
- r_{ij} is the request from $i \in V$ to $j \in V$, with multiplicity m_{ij}. R is the set of all requests.
- $P(G)$ is the set of all simple dipaths in G.
- t stands for a tunnel, and T is the set of tunnels, that is $t \in T \subseteq P(G)$.
- ℓ is a length function on the arcs, that is $\ell : E \to \mathbb{R}^+$.
- for a tunnel t, $\ell(t) = \sum_{e \in t} \ell(e)$ is its length and $w(t)$ is the amount of traffic it carries.

Note that a priori $w(t)$ depends on the routing policy. The cost of a tunnel t is then $w(t) + (\ell(t) - 1)$, and the cost of a set of tunnels T is

$$\sum_{t \in T} (w(t) + \ell(t) - 1)\,. \tag{1}$$

Each tunnel can be seen as a directed hyperarc on the vertex set of G. This observation naturally leads to the definition of a hypergraph layout.

Definition 1 (Hypergraph layout) *Given a graph G and a set $T \subseteq P(G)$, $H(T)$ is the directed hypergraph with $V(H(T)) = V(G)$, and where for each tunnel $t \in T \subseteq P(G)$ there is a directed hyperarc in $H(T)$ connecting any vertex of t to the end of t. $H(T)$ is called a* hypergraph layout*.*

Note that a hypergraph $H(T)$ defines a virtual topology on G. A hypergraph layout $H(T)$ is said to be *feasible* if for each request $r_{ij} \in R$ there exists a dipath in $H(T)$ from i to j. The problem can then be simply expressed as finding a feasible hypergraph layout of minimum cost. Let us now rewrite the cost function of Equation (1).

Given a hypergraph layout $H(T)$, let $L(r_{ij})$ be the number of hyperarcs that request r_{ij} uses, and let $d_H(i,j)$ be the distance from vertex i to vertex j in $H(T)$. Then the term $\sum_{t\in T} w(t)$ of Equation (1) can be rewritten as $\sum_{r_{ij}\in R} L(r_{ij})\cdot m_{ij}$ and, since $L(r_{ij}) \geq d_H(i,j)$, we conclude that in an optimal solution the routing necessarily uses shortest dipaths in the hypergraph layout. It follows that the cost function of Equation (1) can be rewritten w.l.o.g. as

$$\sum_{t\in T}(\ell(t)-1) + \sum_{r_{ij}\in R} d_H(i,j)m_{ij}. \tag{2}$$

The cost of a solution is of bicriteria nature. The first part is the cost of the hypergraph structure; we call it the *total length* of the layout. The second part is the total distance that the traffic travels in the hypergraph; we call it the *total hop count*. Both cost function parts are very much conflicting. On the one hand, to minimize the hop count, it is enough to take a shortest tunnel connecting any source to any destination. On the other hand, to minimize the total length of the layout, it is enough to use a minimum arc-weighted connected hypergraph H such that for each request $r_{ij} \in R$, vertices i and j lie on the same connected component of H. Summarizing, the problem can be stated as follows.

> Minimum Cost Hypergraph Layout: Given a digraph G with a length function and a set R of traffic requests, find a feasible hypergraph layout of minimum cost, where the cost of a hypergraph layout is defined as in Equation (2).

If G is a symmetric digraph, the problem is denoted Minimum Cost Symmetric Hypergraph Layout. It makes sense also to consider the decision version in which we are also given two positive integers C_L, C_H and the objective is to decide whether there exists a layout with total length less than C_L and total hop count less than C_H.

We note that the cost function of Equation (2) can be naturally generalized to

$$\alpha \cdot \sum_{t\in T} c(t) + \beta \cdot \sum_{r_{ij}\in R} d_H(i,j)m_{ij}\ , \tag{3}$$

where α and β are positive constants and $c(t)$ is a general cost function $c : P(G) \to \mathbb{R}^+$. The cost function of Equation (2) corresponds to $c(t) = \ell(t) - 1$.

Relation with VPL problems. This layout design problem defined above is quite similar to well studied VPL problems in ATM networks, in which one imposes a constraint on the logical structure and then wishes to minimize either the maximum distance [2] or the average distance [6] traveled by the traffic. Concerning hardness and approximation, we shall see in the sequel of the article that the problem we study inherits most of the characteristics of the classical VPL problems studied since the 80s. It is not surprising that, even if new technologies like GMPLS are proposed to cope with the increasing bandwidth of communication networks, the computational complexity of the problems associated to these technologies remains essentially the same.

Nevertheless, there are two crucial differences between the GMPLS problem that we study and the classical VPL version of ATM networks. Indeed, we have seen that the GMPLS logical network design problem can be translated into finding a set of dipaths in a *directed hypergraph*, whereas the existing models for VPL problems deal with *digraphs* without multiple arcs. This feature will be exploited in the dynamic programming approach for the path presented in Section 5. The second difference is that the cost function we consider takes into account the *sum* of the length and the hop count costs, whereas usually in VPL problems the aim is to minimize the *maximum* value of either the length or the hop count in the network. Finally, it is important to note that, if there is a single source in the the GMPLS version (or, more generally, if the traffic instance is such that in an optimal solution each hyperarc has exactly 2 vertices), then the problem is basically equivalent to a classical VPL problem.

3 Hardness Results

In this section we give hardness results for the Minimum Cost Hypergraph Layout problem. We distinguish two cases according to whether the underlying network is symmetric or not. We focus on those cases in Sections 3.1 and 3.2.

3.1 General case

Theorem 1 *The* Minimum Cost Hypergraph Layout *problem cannot be approximated within a factor $C \log n$ for some constant $C > 0$, even if the instance is a partial broadcast, unless* P = NP*.*

Proof: The reduction is from Minimum Set Cover[4]. Raz and Safra [10] proved that Minimum Set Cover is not approximable within a factor $C \log n$, for some constant $C > 0$, unless P = NP. To a Set Cover instance with sets $S_1, S_2, \ldots, S_k$, with $S_i \subseteq \{a_1, a_2, \ldots, a_n\}$, we associate the following graph:

- We start with a distinguished node s.

[4] Given a finite set $\mathcal{S}$ and a collection $\mathcal{C}$ of subsets of $\mathcal{S}$, the aim is to find a subcollection $\mathcal{C}'$ of $\mathcal{C}$ of minimum cardinality that covers all the elements of $\mathcal{S}$.

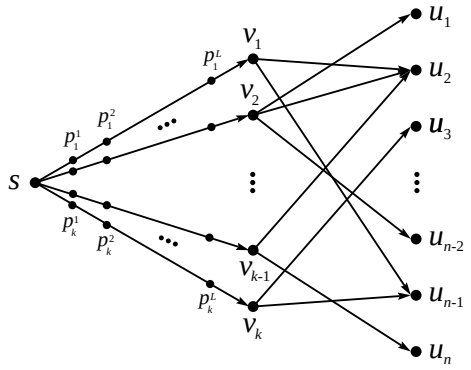

Fig. 2. Reduction in the proof of Theorem 1.

- For each set S_i we introduce a node v_i and a directed path of length $L+1$ (L is a constant to be specified later) from s to v_i through L new vertices $p_i^1, p_i^2, \ldots, p_i^L$.
- For each element a_j we introduce a vertex u_j and, for each vertex v_i we add the arcs (v_i, u_j) if $a_j \in S_i$.
- The requests are from s to u_j, for $i = 1, \ldots, n$.

This construction is illustrated in Fig. 2. Let OPT be the optimal cost to the Minimum Cost Hypergraph Layout instance, and let OPT_{SC} be the optimal cost to the Minimum Set Cover instance.

Note that any cover defined by $I \subseteq \{1, 2, \ldots k\}$ induces a solution of Directed Hypergraph Layout obtained as follows: we use a tunnel of cost L connecting node s to each node $v_i, i \in I$ corresponding to a set taken in the cover. Then we connect each node $v_i, i \in I$ to the vertices $u_j, j \in S_i$. Finally, if a node u_j has more than one incoming tunnel (which means that a_j is covered more than once), we remove extra ones. A solution induced by an optimal cover has length cost $L \cdot OPT_{SC}$, and the hop count cost is $2n$, so $OPT \leq L \cdot OPT_{SC} + 2n$.

Conversely, given a layout, the dipaths from s to v_i used by some tunnel must induce a cover, so $OPT \geq L \cdot OPT_{SC} + n$. Putting all together,

$$L \cdot OPT_{SC} + n \leq OPT \leq L \cdot OPT_{SC} + 2n.$$

By choosing L to be large enough, the gap for the Minimum Cost Hypergraph Layout problem can be made as large as in Minimum Set Cover. Since, unless P = NP, approximating Minimum Set Cover within a factor $C \log n$ for some constant $C > 0$ is NP-hard [10], our result follows. □

3.2 Symmetric case

When the input graph G is symmetric, we can consider G as an undirected where the edge $\{i, j\}$ corresponds to the two arcs (i, j) and (j, i).

Theorem 2 *The* Minimum Cost Symmetric Hypergraph Layout *problem is* APX*-hard even if the instance is a partial broadcast. Therefore, it does not accept a* PTAS *unless* P=NP.

Proof: The reduction is from MINIMUM STEINER TREE[5], which is know to be APX-hard [3], hence it does not accept a PTAS unless P = NP.

Given an instance $(G = (V, E), S \subseteq V)$ of MINIMUM STEINER TREE problem on n vertices, we build an instance of MINIMUM COST SYMMETRIC HYPERGRAPH LAYOUT problem by subdividing $\Omega\left(n^2 \cdot \sum_{r_{ij} \in R} m_{ij}\right)$ times each edge of G and considering as request set a partial broadcast from any vertex in S to all the others vertices in S. Note that subdividing edges is equivalent to setting $\alpha >> \beta$ in the cost function of Equation (3). In other words, the total hop count is negligible compared to the total length of the layout. It is then clear that any optimal solution to the MINIMUM COST SYMMETRIC HYPERGRAPH LAYOUT corresponds to a minimum cost Steiner tree in G spanning all the elements in S. Let OPT be the optimal cost to the MINIMUM COST SYMMETRIC HYPERGRAPH LAYOUT instance, and let OPT_{ST} be the optimal cost to the MINIMUM STEINER TREE instance. Let M be the number of times we have subdivided the edges of G. Summarizing,

$$OPT = M \cdot OPT_{ST} + o(M \cdot OPT_{ST}).$$

The existence of a PTAS for MINIMUM COST SYMMETRIC HYPERGRAPH LAYOUT would yield a PTAS for MINIMUM STEINER TREE, which is impossible unless P = NP. □

4 Approximation Algorithms

In this section we provide approximation algorithms for MINIMUM COST HYPERGRAPH LAYOUT problem. Unless said otherwise, we focus on the symmetric version, for which the description of the algorithms is easier, although the main ideas could be adapted to the general version with slight modifications. For the sake of presentation, we describe our algorithms when the network is a path, a tree, and a general graph in Sections 4.1, 4.2, and 4.3, respectively. The approximation algorithm for the directed path network appeared also in [1], we include it here for the sake of completeness.

4.1 Case of the path

First assume that the instance is a weighted all-to-all (i.e., there is a traffic request between each couple of nodes), and that n is a power of two (otherwise, just add dummy vertices). Then one simply uses the following binary layout: we connect node 1 to node $n/2$, node $n/2$ to node n, and we use recursively the binary layout for $n/2$ on the subdipaths $[1, n/2]$ and $[n/2, n]$. It is clear that any traffic request can be routed in this layout with at most $\log n$ hops, and that the

[5] Given an edge-weighted graph $G = (V, E)$ and a subset $S \subseteq V$, find a connected subgraph with minimum edge-weight containing all the vertices in S. We can assume, by subdividing edges, that all edge-weights equal 1.

total length of this layout is bounded above by $\log n \cdot \ell([1,n])$, where $\ell([1,n])$ denotes the length of the tunnel going from node 1 to node n. Therefore the cost of this layout is $\log n \cdot \sum_{r_{ij} \in R} m_{ij} + \log n \cdot \ell([1,n])$. Since any layout costs at least $\sum_{d \in D} m_{ij} + \ell([1,n])$, this provides a $\log n$-approximation in the all-to-all case.

Now, for a general traffic pattern, it is not always the case that $\ell([1,n])$ is a lower bound on the total length of the layout. We define the *span* of an instance as the minimum set of arcs such that any request can be routed using only those arcs. Note that the span is indeed a set of intervals such that any traffic request is routed within one of these intervals. Let ℓ_0 denote the length of the span. Then any layout costs at least $\sum_{r_{ij} \in R} m_{ij} + \ell_0$, and using the binary layout on each interval of the span we can define a layout with total length $\log n \cdot \ell_0$ and total hop count $\log n \cdot \sum_{r_{ij} \in R} m_{ij}$. Summarizing,

Proposition 1. *When the network is a path, there exists a polynomial-time approximation algorithm for* Minimum Cost Hypergraph Layout *problem with an approximation ratio* $\mathcal{O}(\log n)$.

4.2 Case of the tree

In [2] the authors studied the design of virtual layouts in ATM networks. Their model deals with point-to-point connections in the virtual graph, whereas in Minimum Cost Hypergraph Layout problem, a tunnel can carry more than one request. Nevertheless, we can use the results of [2] to obtain good approximation algorithms. Namely, we are interested in the following result which establishes the trade-off between the maximum load c and the diameter of a virtual layout allowing to route an all-to-all traffic in a general tree.

Theorem 1 (Bermond *et al.* [2]). *In a general tree on n nodes with all-to-all traffic, for each value of $c \in \{1, \ldots, n\}$ there exists a virtual layout allowing to route all traffic with diameter at most $10c \cdot n^{\frac{1}{2c-1}}$ and load at most c. In addition, such a layout can be constructed in polynomial time.*

In particular, if we set $c = \frac{\log n + 1}{2}$, Theorem 1 implies that we can find in polynomial time a layout with load $\mathcal{O}(\log n)$ and diameter at most $(5 \log n + 5) \cdot n^{\frac{1}{\log n}} = 10 \log n + 10 = \mathcal{O}(\log n)$.

Suppose first that the instance of Minimum Cost Hypergraph Layout problem is a weighted all-to-all traffic. It is clear that each arc must be used by some tunnel, hence $n-1$ is a lower bound on the total length of any layout. On the other hand, the hop count is at least $\sum_{r_{ij} \in R} m_{ij}$. In the layout described above, each arc is used at most $\frac{\log n + 1}{2}$ times, and therefore the total length of this layout is $\mathcal{O}(n \log n)$. Since the diameter is also $\mathcal{O}(\log n)$, the total hop count is $\mathcal{O}(\log n \cdot \sum_{r_{ij} \in R} m_{ij})$, yielding an $\mathcal{O}(\log n)$-approximation.

If the instance is not all-to-all, we repeat the argument of the *span* discussed in Section 4.1, obtaining again an $\mathcal{O}(\log n)$-approximation. Summarizing,

Proposition 2. *When the network is a tree, there exists a polynomial-time approximation algorithm for* Minimum Cost Hypergraph Layout *problem with an approximation ratio* $\mathcal{O}(\log n)$.

4.3 General network

In the Minimum Generalized Steiner Network problem, we are given a graph $G = (V, E)$, a weight function $w : E \to \mathbb{N}$, a capacity function $c : E \to \mathbb{N}$, and a requirement function $r : V \times V \to \mathbb{N}$. The objective is to find a *Steiner network* over G that satisfies all the requirements and obeys all the capacities, i.e., a function $f : E \to \mathbb{N}$ such that, for each edge e, $f(e) \leq c(e)$ and, for any pair of nodes i and j, the number of edge disjoint paths between i and j is at least $r(i, j)$, where for each edge e, $f(e)$ copies of e are available. We want to minimize the cost of the network, i.e., $\sum_{e \in E} w(e)f(e)$. The problem is approximable within $\mathcal{O}(\log r_{\max})$, where $r_{\max}$ is the maximum requirement [7], and within a constant factor 2 when all the requirements are equal [8]. The directed version of the problem is approximable within $\mathcal{O}(n^{2/3} \log^{1/3} n)$ [5].

Given an instance of Minimum Cost Hypergraph Layout in a general network, consider the associated Minimum Generalized Steiner Network problem where all the requirements are equal to 1 and where the edge capacities are set to $+\infty$. Let H be an optimal solution to this Minimum Generalized Steiner Network instance (note that H may be disconnected). The following easy observation will be useful: since H is the smallest subgraph of G such that any couple source-destination lies on the same connected component, in any solution to the Minimum Cost Hypergraph Layout problem, the number of arcs that are used by at least one tunnel is at least $|E(H)|$. Using the algorithm of [8], we can find in polynomial time a Steiner network H' with $|E(H')| \leq 2|E(H)|$. Since the edge capacities are set to ∞, we can assume that such a Steiner network is a forest. The layout is then obtained by applying the algorithm described in Section 4.2 to each connected component of H'.

It is clear that the hop count of this layout is at most $\mathcal{O}(\log n)$ times the lower bound $\sum_{r_{ij} \in R} m_{ij}$. On the other hand, the total length of this layout is $\mathcal{O}(\log n \cdot |E(H')|) = \mathcal{O}(\log n \cdot |E(H)|)$. Since the total length of any layout is lower-bounded by $|E(H)|$, the $\mathcal{O}(\log n)$-approximation follows. Summarizing,

Theorem 2. *In a general network, there exists a polynomial-time approximation algorithm for* Minimum Cost Hypergraph Layout *problem with an approximation ratio* $\mathcal{O}(\log n)$.

5 The Hypergraph Layout Problem on the Path

In this section we focus on the case when the underlying digraph is a directed path (nodes are numbered from left to right $1, \ldots, n$). Our approach consists in a dynamic programming algorithm that computes partial solutions induced on

subdipaths of the original path. We denote by $[i, j]$ the subpath from node i to node j.

Loosely speaking, we use the following dynamic program: we consider a cut vertex i and we look at a local solution induced on the subpath $[1, i]$. That is, the tunnels and traffic located on $[1, i]$. The cost of a local solution is defined as the sum of the local tunnels cost plus the hop counts sum taken on the local traffic.

We introduce then node $i+1$ and the potential tunnels finishing at it. In order to update the local solution cost, it is necessary to have enough information to compute the hop counts once this tunnel is introduced in the solution. So for each source $s \in S$ and vertex x, we introduce $h(s, x)$ defined as the hop count from s to x. Each vertex is then characterized by a hop count vector $h(x)$ whose dimension is the number of sources. A partial solution is then fully encoded by its local cost and the hop counts of all its nodes. It follows from the above discussion that we can encode a partial solution by giving, for each of its hop count vectors, the rightmost node associated to that vector. If we denote by h a bound on the hop count (at most n) and by c a bound (at most n) on the tunnel cost, we have $(c^k)^{h^k} = c^{kh^k}$ such possible table entries, where k is the number of sources.

By making an error of ε on the two costs (length and hops), we can encode the logarithm in base $1 + \varepsilon$ of those quantities, which leads to tables of size $\Theta\left((\log n)^{k(\log n)^k}\right)$. Note that this running time is already subexponential, so the problem is unlikely to be NP-hard to approximate within a constant factor when the number of sources is bounded (because it is widely assumed that algorithms solving 3-SAT require $2^{\Theta(n)}$ time). We shall see now how to improve this first naïve dynamic program.

We proceed now to give all the details for one and two sources, that suffice to get the intuition for an arbitrary number k of sources.

5.1 Case of a single source and the non crossing property

We summarize the algorithm that appeared first in [1] (Gerstel *et al.* used a similar approach in [6]). In the case of a single source, it is not difficult to see that the tunnel structure is *non crossing*, i.e. two tunnels can only intersect in an optimal solution if one is strictly inside the other [1]. Since the path is directed, we assume w.l.o.g. that the source is located in the leftmost node of the path. This leads to the following approach: we consider the rightmost tunnel originating from the source and assume it ends at node i. Clearly, any tunnel starting in $[1, i-1]$ and ending in $[i, n]$ can be replaced with a tunnel starting at i, since this new tunnel may only decrease the hop count and the length[6].

This approach allows us to compute the optimal for a path with n vertices inductively. We denote by $C[i, j]$ the minimum cost for the requests destined to

[6] this fact holds for any increasing cost function $c(t)$ in Equation (3).

the subdipath $[i, j]$, in which the source is replaced by node i. Then for $2 \leq i \leq n$,

$$C[1,i] = \min_{k<i} \left\{ C[1,k-1] + \left(\sum_{e \in E([1,k])} \ell(e) - 1 \right) + \sum_{j=k}^{i} m_{1j} + C[k,i] \right\}. \quad (4)$$

Note that $C[1,n]$ is the optimal cost of the original problem, and it can be computed in time $\mathcal{O}(n^3)$ [1]. In the particular case of the uniform broadcast (that is, $m_{ij} = 1$ for $i = 1$ and $2 \leq j \leq n$, and $m_{ij} = 0$ otherwise) and with a unitary length function on the edges, a closed formula was given in [1].

5.2 Case of two sources

We use a dynamic program similar to the one used for the single source case, but slightly more complicated. Let s_0 be the leftmost source and let s_1 be the other. In order to solve the problem, we introduce an auxiliary problem with *pseudo-sources*. A pseudo-source s is denoted by a triple (h_0, h_1, l), where l is the distance from s to the subdipath that lies on the right of the rightmost pseudo-source, and where h_i indicates that from s one can reach s_i in h_i hops, $i = 0, 1$. In the induction of the dynamic program the following auxiliary problem will appear:

- The traffic is restricted to an interval $[u, v]$, where either u or v is an end of the original dipath.
- There are one or two pseudo-sources located to the left of $[u, v]$.
- If there are two pseudo-sources, they are labeled (j, j, l_0) and $(j+1, j, l_1)$, and we denote the corresponding problem $P((j,j),(j+1,j),l_0,l_1,[u,v])$.
- If there is a single pseudo-source, it is labeled (j, k), and we denote the problem $P((j,k),[u,v])$.

In both cases we denote by OPT the cost of an optimal solution. Note that $P((0,0),[u,v])$ is indeed a single source problem in which a unique source replaces both s_0 and s_1. Moreover, $P((j,k),[u,v])$ is equivalent to a single source problem, since $OPT(P((j,k),[u,v])) = OPT\left(P(0,0,[u,v])) + \sum_{x \in [u,v]} (j \cdot m_{s_0 x} + k \cdot m_{s_1 x})\right)$. We now relate the two sources problem to the auxiliary problem. Consider the rightmost tunnel having s_i as head and denote by E_i its end node, $i = 0, 1$. We compute an optimal solution conditioned to the values E_i, $i = 0, 1$. There are three cases to consider, as it is depicted in Fig. 3.

(a) E_0 is left to s_1. Then on the subpath $[E_0, n]$ we pick an optimal solution with a slightly modified instance: we leave traffic requests toward s_1 unchanged and we replace the source s_0 by a pseudo-source at E_0 with hop count 1. On the subpath $[s_0, E_0 - 1]$ we use an optimal solution (note that in this subproblem there is only one source).

(b) E_0 is on the right of both s_1 and E_1. Then E_0 is at distance 1 from both sources and therefore any tunnel entering $[E_0 + 1, n]$ can be assumed to start at E_0. So the optimal solution is then obtained by using $OPT([s_0, E_0])$.

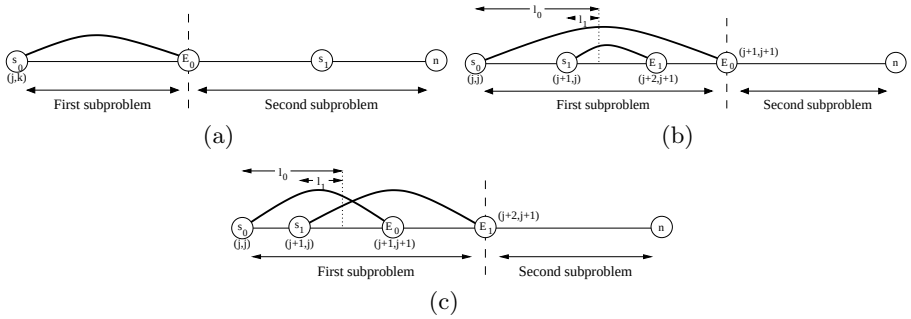

Fig. 3. Dynamic programming with two sources: cases (a), (b), and (c), respectively.

Note that in both cases (a) and (b) the induction is valid because E_0 is the best node to start a tunnel going to its right. Indeed, starting at E_0 is cheaper, and no node closer to the sources can be reached (from the definition of E_0).

(c) E_1 on the right of E_0. Note that E_0 is a $(1,1)$ pseudo-source, while E_1 is a $(2,1)$ pseudo-source. Consider a tunnel ending in $[E_1+1, n]$. The situation gets more complicated than in the single source case, since E_1 (a $(2,1)$ node) is not the "best" possible node anymore. The only nodes that can beat E_1 are $(1,1)$ nodes, and E_0 is the rightmost one. Hence we can assume that such a tunnel is starting either at E_0 or at E_1. Indeed, we have two "best nodes". So to perform the induction we have to solve two subproblems:

(c.1) the first subproblem on $[s_0, E_1 - 1]$, but under a condition on the location of the rightmost tunnel from s_1, i.e. $OPT([s_0, E_1 - 1] \mid (s_1, E_1))$.

(c.2) the second subproblem in which we have two pseudo-sources E_0 of type $(1,1)$ and E_1 of type $(2,1)$. So we pay $OPT(P((1,1),(2,1), l(E_0, E_1), l(E_1, E_1 + 1), [E_1 + 1, n]))$.

To complete our algorithm we need to show how to compute the dynamic program tables inductively, i.e. to compute $OPT(P((j,j),(j+1,j), l_0, l_1, [u,v]))$.

The two pseudo-sources tables. The induction is again on the two rightmost nodes E_0, E_1, with essentially the same cases as above, except case (a), which cannot occur since both pseudo-sources are now located outside the path.

(i) E_0 is on the right of E_1. Then E_0 is at distance $j+1$ from both sources and therefore any tunnel entering $[E_0 + 1, n]$ can be assumed to start at E_0. So the optimal solution is obtained by using $OPT([s_0, E_0])$ for the second subproblem and $OPT(P((j,j),(j+1,j), l_0, l_1, [s_0, E_0 - 1]))$.

(ii) E_0 is on the left of E_1. The situation is similar to case (c). We can split the problem into two subproblems, the first one being a two pseudo-sources problem reduced to $[u, E_1 - 1]$ with a condition on the rightmost tunnel from s_0, and the second being a single source problem on $[E_1 + 1, n]$.

Correctness & complexity. To complete the proof, we must explain how the above induction allows to compute all the tables inductively. Here are some explanations:

- First the induction is performed on the length of the path and when the tables for $[u, v]$ are computed, all the tables for strict subdipath of $[u, v]$ are known.
- Second, when filling the new tables, we compute the cost in a consistent way: we sum the cost of the first and second subproblems (found in already computed tables) with the cost of the tunnels that are removed, plus the hop count for traffic toward the removed node (either E_0 or E_1).
- As usual we keep only the best cost found when examining all the subcases 1,2,3.
- Finally, one may worry about the conditioning on the rightmost tunnel that appears in case (c). But fortunately this never leads to a condition on an unbounded number of tunnels, since in the induction those rightmost tunnels either disappear or stay.

To evaluate the complexity we use a pessimistic bound on the table size, $OPT(P((j, j), (j + 1, j), l_0, l_1, [u, v]))$. The values of l_0, l_1 are polynomial since they are in bijection with the pseudo-sources locations, $j \in [1, n]$. Since $[u, v]$ is either an end or a head segment we can store it in space $2n$. Therefore we get size $\Theta(n^4)$ for the tables, and if we add the conditioning on the rightmost tunnel from the rightmost source we get $\Theta(n^5)$.

Finally, to improve the complexity we can use classic scaling technics to get space $\frac{\log n}{\varepsilon}^5$ and approximation factor $1 + \varepsilon$.

6 Conclusions and Further Research

In this paper we modeled a question raised by label minimization in GMPLS networks as a hypergraph layout problem. In the single commodity case we showed the problem to be closely related to well studied VPL problems. However, the optimization criteria (average hop count and average load) that appear in our problem are ones of the less studied. We observed that approximation results follow immediately from extension of the results known for fixed depth hierarchical facility location (equivalently, bounded depth metric Steiner trees) to the average depth case. We also gave a general $\log n$-approximation that is universal (that is, it does not depend on the traffic), as well as hardness results.

In the multi-sources case, we presented a dynamic program on the path that is polynomial when the number of sources is fixed. So finding a polynomial algorithm in the general case on the path remains open; likely extensions of the dynamic program to the case of trees and bounded treewidth networks remain to be done. Last, we believe that more general approximation results can be given for low dimension Euclidean metric graphs using the classical Arora paradigm.

References

1. J.-C. Bermond, D. Coudert, J. Moulierac, S. Perennes, H. Rivano, I. Sau, and F. Solano Donado. MPLS label stacking on the line network. In *Proc. of IFIP Networking*, volume 5550 of *LNCS*, pages 809–820, 2009.
2. J.-C. Bermond, N. Marlin, D. Peleg, and S. Pérennes. Directed virtual path layouts in ATM networks. *Theoretical Computer Science*, 291(1):3–28, 2003.
3. M. Bern and P. Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32:171–176, 1989.
4. S. Bhatnagar, S. Ganguly, and B. Nath. Creating Multipoint-to-Point LSPs for traffic engineering. *IEEE Commun. Mag.*, 43(1):95–100, Jan. 2005.
5. M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed Steiner problems. In *Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 192–200, 1998.
6. O. Gerstel, A. Wool, and S. Zaks. Optimal layouts on a chain ATM network. *Discrete Applied Mathematics*, 83:157–178, 1998.
7. M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, E. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. In *Proc. of the 5th annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 223–232, 1994.
8. S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41:214–235, 1994.
9. F. Ramos et al. IST-LASAGNE: Towards all-optical label swapping employing optical logic gates and optical flip-flops. *IEEE J. Sel. Areas Commun.*, 23(10):2993–3011, 2005.
10. R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. of the 29th annual ACM Symposium on Theory of Computing (STOC)*, pages 475–484, 1997.
11. H. Saito, Y. Miyao, and M. Yoshida. Traffic engineering using multiple MultiPoint-to-Point LSPs. In *Proc. of IEEE INFOCOM*, pages 894–901, 2000.
12. F. Solano, R. V. Caenegem, D. Colle, J. L. Marzo, M. Pickavet, R. Fabregat, and P. Demeester. All-optical label stacking: Easing the trade-offs between routing and architecture cost in all-optical packet switching. In *Proc. of IEEE INFOCOM*, pages 655–663, 2008.
13. F. Solano, R. Fabregat, and J. Marzo. On optimal computation of MPLS label binding for MultiPoint-to-Point connections. *IEEE Trans. Commun.*, 56(7):1056–1059, 2007.
14. F. Solano and J. Moulierac. Routing in All-Optical Label Switched-based Networks with Small Label Spaces. In *Proc. of the 13th Conference on Optical Network Design and Modeling (ONDM)*, 2009.
15. F. Solano, T. Stidsen, R. Fabregat, and J. Marzo. Label space reduction in MPLS networks: How much can one label do? *IEEE/ACM Trans. Netw.*, 2009.

Appendix C

Tolerance Graphs

C.1 A New Intersection Model and Improved Algorithms

C.2 The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete

A New Intersection Model and Improved Algorithms for Tolerance Graphs*

George B. Mertzios† Ignasi Sau‡ Shmuel Zaks§

Abstract

Tolerance graphs model interval relations in such a way that intervals can tolerate a certain degree of overlap without being in conflict. This class of graphs, which generalizes in a natural way both interval and permutation graphs, has attracted many research efforts since their introduction in [10], as it finds many important applications in constraint-based temporal reasoning, resource allocation, and scheduling problems, among others. In this article we propose the first non-trivial intersection model for general tolerance graphs, given by three-dimensional parallelepipeds, which extends the widely known intersection model of parallelograms in the plane that characterizes the class of bounded tolerance graphs. Apart from being important on its own, this new representation also enables us to improve the time complexity of three problems on tolerance graphs. Namely, we present optimal $\mathcal{O}(n \log n)$ algorithms for computing a minimum coloring and a maximum clique, and an $\mathcal{O}(n^2)$ algorithm for computing a maximum weight independent set in a tolerance graph with n vertices, thus improving the best known running times $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ for these problems, respectively.

Keywords. Tolerance graphs, parallelogram graphs, intersection model, minimum coloring, maximum clique, maximum weight independent set.

AMS subject classification (2000). Primary 05C62; Secondary 05C85, 05C15, 05C69, 68R10.

1 Introduction

A graph $G = (V, E)$ on n vertices is a *tolerance graph* if there is a set $I = \{I_i \mid i = 1, \ldots, n\}$ of closed intervals on the real line and a set $T = \{t_i \mid i = 1, \ldots, n\}$ of positive real numbers, called *tolerances*, such that for any two vertices $v_i, v_j \in V$, $v_i v_j \in E$ if and only if $|I_i \cap I_j| \geq \min\{t_i, t_j\}$, where $|I|$ denotes the length of the interval I. These sets of intervals and tolerances form a *tolerance representation* of G. If G has a tolerance representation such that $t_i \leq |I_i|$ for $i = 1, \ldots, n$, then G is called a *bounded tolerance graph* and its representation is a *bounded tolerance representation.*

Tolerance graphs were introduced in [10], mainly motivated by the need to solve scheduling problems in which resources that would be normally used exclusively, like rooms or vehicles, can tolerate some sharing among users. Since then, tolerance graphs have been widely studied in the literature [1, 2, 5, 11, 12, 15, 19, 22], as they naturally generalize both interval graphs (when all tolerances are equal) and permutation graphs (when $|I_i| = t_i$ for $i = 1, \ldots, n$) [10]. For more details, see [13].

*A preliminary conference version of this work will be presented in the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), Montpellier, France, June 24-26, 2009.

†Department of Computer Science, RWTH Aachen, Germany. Email: mertzios@cs.rwth-aachen.de

‡Mascotte joint Project of INRIA/CNRS/UNSA, Sophia-Antipolis, France; and Graph Theory and Combinatorics Group, Applied Maths. IV Dept. of UPC, Barcelona, Spain. Email: ignasi.sau@sophia.inria.fr

§Department of Computer Science, Technion, Haifa, Israel. Email: zaks@cs.technion.ac.il

Notation. All the graphs considered in this paper are finite, simple, and undirected. Given a graph $G = (V, E)$, we denote by n the cardinality of V. An edge between vertices u and v is denoted by uv, and in this case vertices u and v are said to be *adjacent*. $\overline{G}$ denotes the *complement* of G, i.e. $\overline{G} = (V, \overline{E})$, where $uv \in \overline{E}$ if and only if $uv \notin E$. Given a subset of vertices $S \subseteq V$, the graph $G[S]$ denotes the graph *induced* by the vertices in S, i.e. $G[S] = (S, F)$, where for any two vertices $u, v \in S$, $uv \in F$ if and only if $uv \in E$. A subset $S \subseteq V$ is an *independent set* in G if the graph $G[S]$ has no edges. For a subset $K \subseteq V$, the induced subgraph $G[K]$ is a *complete subgraph* of G, or a *clique*, if each two of its vertices are adjacent (equivalently, K is an independent set in $\overline{G}$). The maximum cardinality of a clique in G is denoted by $\omega(G)$ and is termed the *clique number* of G. A *proper coloring* of G is an assignment of different colors to adjacent vertices, which results in a partition of V into independent sets. The minimum number of colors for which there exists a proper coloring is denoted by $\chi(G)$ and is termed the *chromatic number* of G. A partition of V into $\chi(G)$ independent sets is a *minimum coloring* of G.

Motivation and previous work. Besides generalizing interval and permutation graphs in a natural way, the class of tolerance graphs has other important subclasses and superclasses. Let us briefly survey some of them.

Given a class of graphs $\mathcal{G}$, a graph G is a *probe graph* of $\mathcal{G}$ if its vertices can be partitioned into a set $\mathbb{P}$ of probes and an independent set $\mathbb{N}$ of nonprobes, such that G can be extended to a graph of $\mathcal{G}$ by adding edges between certain nonprobes. Probe interval graphs, which are a superclass of interval graphs, have proved to be useful in the Human Genome Project [23]. Tolerance graphs are also a superclass of interval probe graphs [12].

A graph is *perfect* if the chromatic number of every induced subgraph equals the clique number of that subgraph. Perfect graphs include many important families of graphs, and serve to unify results relating colorings and cliques in those families. For instance, in all perfect graphs, the graph coloring problem, maximum clique problem, and maximum independent set problem can all be solved in polynomial time using the Ellipsoid method [14]. Since tolerance graphs were shown to be perfect [11], there exist polynomial time algorithms for these problems. However, these algorithms are not very efficient and therefore, as it happens for most known subclasses of perfect graphs, it makes sense to devise specific fast algorithms for these problems on tolerance graphs.

A *comparability* graph is a graph which can be transitively oriented. A *co-comparability* graph is a graph whose complement is a comparability graph. Bounded tolerance graphs are co-comparability graphs [10], and therefore all known polynomial time algorithms for co-comparability graphs apply to bounded tolerance graphs. This is one of the main reasons why for many problems the existing algorithms have better running time in bounded tolerance graphs than in general tolerance graphs.

A graph $G = (V, E)$ is the *intersection graph* of a family $F = \{S_1, \ldots, S_n\}$ of distinct nonempty subsets of a set S if there exists a bijection $\mu : V \to F$ such that for any two distinct vertices $u, v \in V$, $uv \in E$ if and only if $\mu(u) \cap \mu(v) \neq \emptyset$. In that case, we say that F is an *intersection model* of G. It is easy to see that each graph has a trivial intersection model based on adjacency relations [21]. Some intersection models provide a natural and intuitive understanding of the structure of a class of graphs, and turn out to be very helpful to find efficient algorithms to solve optimization problems [21]. Therefore, it is of great importance to establish non-trivial intersection models for families of graphs. A graph G on n vertices is a *parallelogram graph* if we can fix two parallel lines L_1 and L_2, and for each vertex $v_i \in V(G)$ we can assign a parallelogram $\overline{P}_i$ with parallel sides along L_1 and L_2 so that G is the intersection graph of $\{\overline{P}_i \mid i = 1, \ldots, n\}$. It was proved in [1, 20] that a graph is a bounded tolerance graph if and only if it is a parallelogram graph. This characterization provides a useful way to think about bounded tolerance graphs. However, this intersection model cannot cope with general tolerance graphs, in which the tolerance of an interval can be greater than its length.

Our contribution. In this article we present the first non-trivial intersection model for general tolerance graphs, which generalizes the widely known parallelogram representation of bounded tolerance graphs. The main idea is to exploit the third dimension to capture the information given by unbounded tolerances, and as a result parallelograms are replaced with parallelepipeds. The proposed intersection model is very intuitive and can be efficiently constructed from a tolerance representation (actually, we show that it can be constructed in linear time).

Apart from being important on its own, this new representation proves to be a powerful tool for designing efficient algorithms for general tolerance graphs. Indeed, using our intersection model we improve the best existing running times of three problems on tolerance graphs. We present algorithms to find a minimum coloring and a maximum clique in $\mathcal{O}(n \log n)$ time, which is optimal (as discussed in Section 3.4). The best existing algorithm was $\mathcal{O}(n^2)$ [12, 13]. We also present an algorithm to find a maximum weight independent set in $\mathcal{O}(n^2)$ time, whereas the best known algorithm was $\mathcal{O}(n^3)$ [13]. We note that [22] proposes an $\mathcal{O}(n^2 \log n)$ algorithm to find a maximum *cardinality* independent set on a general tolerance graph, and that [13] refers to an algorithm transmitted by personal communication with running time $\mathcal{O}(n^2 \log n)$ to find a maximum weight independent set on a general tolerance graph; to the best of our knowledge, this algorithm has not been published.

It is important to note that the complexity of recognizing bounded and general tolerance graphs is a challenging open problem [3,13,22], and this is the reason why we assume throughout this paper that along with the input tolerance graph we are also given a tolerance representation of it. The only "positive" result in the literature concerning recognition of tolerance graphs is a linear time algorithm for the recognition of bipartite tolerance graphs [3].

Nevertheless, it was shown in [15] that every tolerance graph has a polynomial sized tolerance representation, and hence tolerance graphs recognition is in the class NP. There exist other graph classes closely related to tolerance graphs. If in the definition of tolerance graphs we replace the operation "min" between tolerances with "+", we obtain the class of *sum-tolerance graphs* [17], and if we replace it with "max" we obtain the class of *max-tolerance graphs*. Max-tolerance graphs recognition is known to be NP-hard [18].

Organization of the paper. We provide the new intersection model of general tolerance graphs in Section 2. In Section 3 we present a canonical representation of tolerance graphs, and then show how it can be used in order to obtain optimal $\mathcal{O}(n \log n)$ algorithms for finding a minimum coloring and a maximum clique in a tolerance graph. In Section 4 we present an $\mathcal{O}(n^2)$ algorithm for finding a maximum weight independent set. Finally, Section 5 is devoted to conclusions and open problems.

2 A New Intersection Model for Tolerance Graphs

One of the most natural representations of bounded tolerance graphs is given by parallelograms between two parallel lines in the Euclidean plane [1,13,20]. In this section we extend this representation to a three-dimensional representation of general tolerance graphs.

Given a tolerance graph $G = (V, E)$ along with a tolerance representation of it, recall that vertex $v_i \in V$ corresponds to an interval $I_i = [a_i, b_i]$ on the real line with a tolerance $t_i \geq 0$. W.l.o.g. we may assume that $t_i > 0$ for every vertex v_i [13].

Definition 1 *Given a tolerance representation of a tolerance graph $G = (V, E)$, vertex v_i is* bounded *if $t_i \leq |I_i|$. Otherwise, v_i is* unbounded. *V_B and V_U are the sets of bounded and unbounded vertices in V, respectively. Clearly $V = V_B \cup V_U$.*

We can also assume w.l.o.g. that $t_i = \infty$ for any unbounded vertex v_i, since if v_i is unbounded, then the intersection of any other interval with I_i is strictly smaller than t_i. Let L_1 and L_2 be two parallel lines at distance 1 in the Euclidean plane.

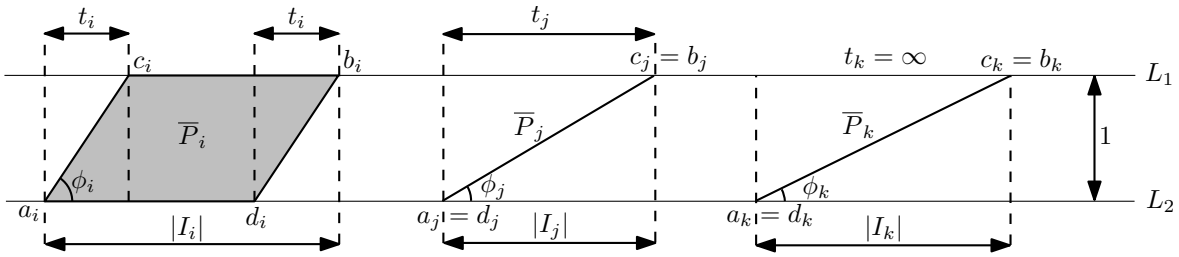

Figure 1: Parallelograms $\overline{P}_i$ and $\overline{P}_j$ correspond to bounded vertices v_i and v_j, respectively, whereas $\overline{P}_k$ corresponds to an unbounded vertex v_k.

Definition 2 *Given an interval $I_i = [a_i, b_i]$ with tolerance t_i, $\overline{P}_i$ is the parallelogram defined by the points c_i, b_i in L_1 and a_i, d_i in L_2, where $c_i = \min\{b_i, a_i + t_i\}$ and $d_i = \max\{a_i, b_i - t_i\}$. The* slope ϕ_i *of $\overline{P}_i$ is $\phi_i = \arctan\left(\frac{1}{c_i - a_i}\right)$.*

An example is depicted in Figure 1, where $\overline{P}_i$ and $\overline{P}_j$ correspond to bounded vertices v_i and v_j, and $\overline{P}_k$ corresponds to an unbounded vertex v_k. Observe that when vertex v_i is bounded, the values c_i and d_i coincide with the *tolerance points* defined in [7, 13, 16], and $\phi_i = \arctan\left(\frac{1}{t_i}\right)$. On the other hand, when vertex v_i is unbounded, the values c_i and d_i coincide with the endpoints b_i and a_i of I_i, respectively, and $\phi_i = \arctan\left(\frac{1}{|I_i|}\right)$. Observe also that in both cases $t_i = b_i - a_i$ and $t_i = \infty$, parallelogram $\overline{P}_i$ is reduced to a line segment (c.f. $\overline{P}_j$ and $\overline{P}_k$ in Figure 1). Since $t_i > 0$ for every vertex v_i, it follows that $0 < \phi_i < \frac{\pi}{2}$. Furthermore, we can assume w.l.o.g. that all points a_i, b_i, c_i, d_i and all slopes ϕ_i are distinct [7, 13, 16].

Observation 1 *Let $v_i \in V_U, v_j \in V_B$. Then $|I_i| < t_j$ if and only if $\phi_i > \phi_j$.*

We are ready to give the main definition of this article.

Definition 3 *Let $G = (V, E)$ be a tolerance graph with a tolerance representation $\{I_i = [a_i, b_i], t_i \mid i = 1, \ldots, n\}$. For every $i = 1 \ldots, n$, P_i is the parallelepiped in $\mathbb{R}^3$ defined as follows:*

(a) If $t_i \leq b_i - a_i$ (that is, v_i is bounded), then $P_i = \{(x, y, z) \in \mathbb{R}^3 \mid (x, y) \in \overline{P}_i,\ 0 \leq z \leq \phi_i\}$.

(b) If $t_i > b_i - a_i$ (v_i is unbounded), then $P_i = \{(x, y, z) \in \mathbb{R}^3 \mid (x, y) \in \overline{P}_i,\ z = \phi_i\}$.

The set of parallelepipeds $\{P_i \mid i = 1, \ldots, n\}$ is a parallelepiped representation *of G.*

Observe that for each interval I_i, the parallelogram $\overline{P}_i$ of Definition 2 (see also Figure 1) coincides with the projection of the parallelepiped P_i on the plane $z = 0$. An example of the construction of these parallelepipeds is given in Figure 2, where a set of eight intervals with their associated tolerances is given in Figure 2(a). The corresponding tolerance graph G is depicted in Figure 2(b), while the parallelepiped representation is illustrated in Figure 2(c). In the case $t_i < b_i - a_i$, the parallelepiped P_i is three-dimensional, c.f. P_1, P_3, and P_5, while in the border case $t_i = b_i - a_i$ it degenerates to a two-dimensional rectangle, c.f. P_7. In these two cases, each P_i corresponds to a bounded vertex v_i. In the remaining case $t_i = \infty$ (that is, v_i is unbounded), the parallelepiped P_i degenerates to a one-dimensional line segment above plane $z = 0$, c.f. P_2, P_4, P_6, and P_8.

We prove now that these parallelepipeds form a three-dimensional intersection model for the class of tolerance graphs (namely, that every tolerance graph G can be viewed as the intersection graph of the corresponding parallelepipeds P_i).

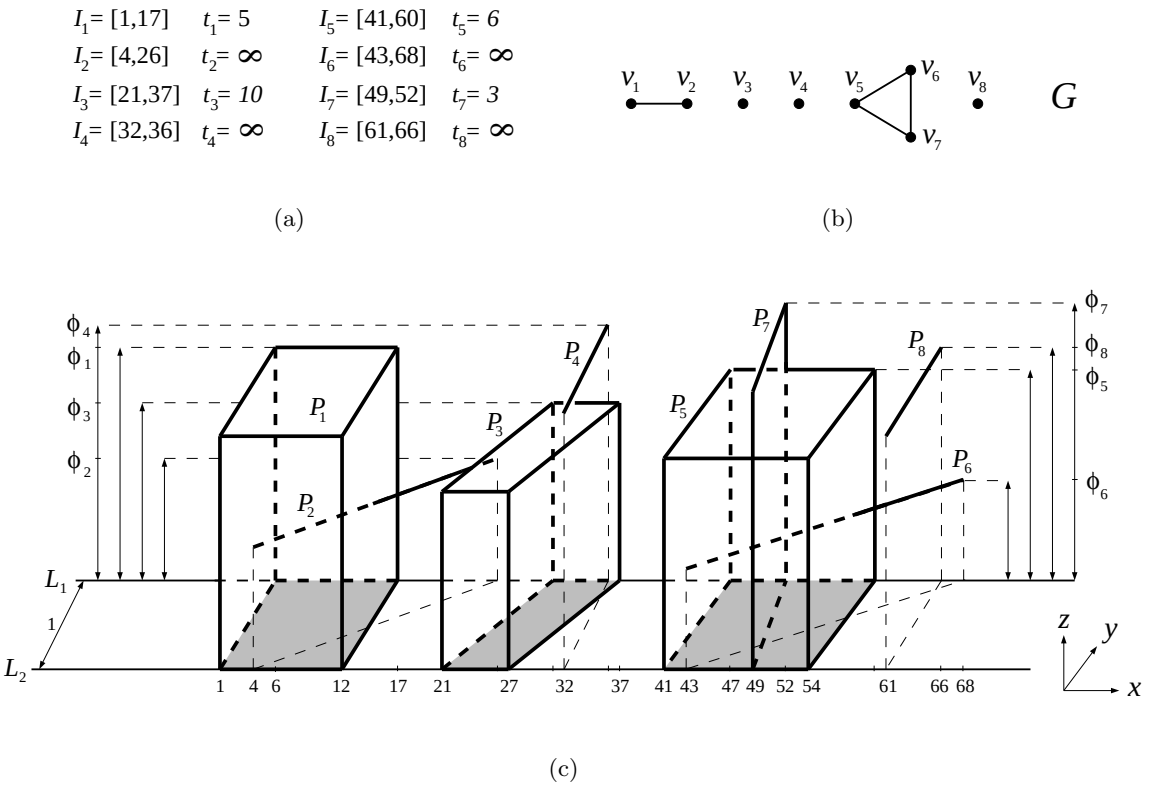

Figure 2: The intersection model for tolerance graphs: (a) a set of intervals $I_i = [a_i, b_i]$ and tolerances t_i, $i = 1, \ldots, 8$, (b) the corresponding tolerance graph G and (c) a parallelepiped representation of G.

Theorem 1 *Let $G = (V, E)$ be a tolerance graph with a tolerance representation $\{I_i = [a_i, b_i], t_i \mid i = 1, \ldots, n\}$. Then for every $i \neq j$, $v_i v_j \in E$ if and only if $P_i \cap P_j \neq \emptyset$.*

Proof. We distinguish three cases according to whether vertices v_i and v_j are bounded or unbounded:

(a) Both vertices are bounded, that is $t_i \leq b_i - a_i$ and $t_j \leq b_j - a_j$. It follows from [13] that $v_i v_j \in E(G)$ if and only if $\overline{P}_i \cap \overline{P}_j \neq \emptyset$. However, due to the definition of the parallelepipeds P_i and P_j, in this case $P_i \cap P_j \neq \emptyset$ if and only if $\overline{P}_i \cap \overline{P}_j \neq \emptyset$ (c.f. P_1 and P_3, or P_5 and P_7, in Figure 2).

(b) Both vertices are unbounded, that is $t_i = t_j = \infty$. Since no two unbounded vertices are adjacent, $v_i v_j \notin E(G)$. On the other hand, the line segments P_i and P_j lie on the disjoint planes $z = \phi_i$ and $z = \phi_j$ of $\mathbb{R}^3$, respectively, since we assumed that the slopes ϕ_i and ϕ_j are distinct. Thus, $P_i \cap P_j = \emptyset$ (c.f. P_2 and P_4).

(c) One vertex is unbounded (that is, $t_i = \infty$) and the other is bounded (that is, $t_j \leq b_j - a_j$). If $\overline{P}_i \cap \overline{P}_j = \emptyset$, then $v_i v_j \notin E$ and $P_i \cap P_j = \emptyset$ (c.f. P_1 and P_6). Suppose that $\overline{P}_i \cap \overline{P}_j \neq \emptyset$. We distinguish two cases:

(i) $\phi_i < \phi_j$. It is easy to check that $|I_i \cap I_j| \geq t_j$ and thus $v_i v_j \in E$. Since $\overline{P}_i \cap \overline{P}_j \neq \emptyset$ and $\phi_i < \phi_j$, then necessarily the line segment P_i intersects with the parallelepiped P_j on the plane $z = \phi_i$, and thus $P_i \cap P_j \neq \emptyset$ (c.f. P_1 and P_2).

(ii) $\phi_i > \phi_j$. Clearly $|I_i \cap I_j| < t_i = \infty$. Furthermore, since $\phi_i > \phi_j$, Observation 1 implies that $|I_i \cap I_j| \leq |I_j| < t_i$. It follows that $|I_i \cap I_j| < \min\{t_i, t_j\}$, and thus $v_i v_j \notin E$. On the other hand, $z = \phi_i$ for all points $(x, y, z) \in P_i$, while $z \leq \phi_j < \phi_i$ for all points $(x, y, z) \in P_j$, and therefore $P_i \cap P_j = \emptyset$ (c.f. P_3 and P_4). ■

Clearly, for each $v_i \in V$ the parallelepiped P_i can be constructed in constant time. Therefore,

Lemma 1 *Given a tolerance representation of a tolerance graph G with n vertices, a parallelepiped representation of G can be constructed in $\mathcal{O}(n)$ time.*

3 Coloring and Clique Algorithms in $\mathcal{O}(n \log n)$

In this section we present optimal $\mathcal{O}(n \log n)$ algorithms for constructing a minimum coloring and a maximum clique in a tolerance graph $G = (V, E)$ with n vertices, given a parallelepiped representation of G. These algorithms improve the best known running time $\mathcal{O}(n^2)$ of these problems on tolerance graphs [12, 13]. First, we introduce a canonical representation of tolerance graphs in Section 3.1, and then we use it to obtain the algorithms for the minimum coloring and the maximum clique problems in Section 3.2. Finally, we discuss the optimality of both algorithms in Section 3.4.

3.1 A canonical representation of tolerance graphs

We associate with every vertex v_i of G the point $p_i = (x_i, y_i)$ in the Euclidean plane, where $x_i = b_i$ and $y_i = \frac{\pi}{2} - \phi_i$. Since all endpoints of the parallelograms $\overline{P}_i$ and all slopes ϕ_i are distinct, all coordinates of the points p_i are distinct as well. Similarly to [12, 13], we state the following two definitions.

Definition 4 *An unbounded vertex $v_i \in V_U$ of a tolerance graph G is called* inevitable *(for a certain parallelepiped representation), if replacing P_i with $\{(x, y, z) \mid (x, y) \in P_i, 0 \leq z \leq \phi_i\}$ creates a new edge in G. Otherwise, v_i is called* evitable.

Definition 5 *Let $v_i \in V_U$ be an inevitable unbounded vertex of a tolerance graph G (for a certain parallelepiped representation). A vertex v_j is called a* hovering vertex *of v_i if $a_j < a_i$, $b_i < b_j$, and $\phi_i > \phi_j$.*

It is now easy to see that, by Definition 5, if v_j is a hovering vertex of v_i, then $v_i v_j \notin E$. Note that, in contrast to [12], in Definition 4, an isolated vertex v_i might be also inevitable unbounded, while in Definition 5, a hovering vertex might be also unbounded. Definitions 4 and 5 imply the following lemma:

Lemma 2 *Let $v_i \in V_U$ be an inevitable unbounded vertex of the tolerance graph G (for a certain parallelepiped representation). Then, there exists a hovering vertex v_j of v_i.*

Proof. Since v_i is an inevitable unbounded vertex, replacing P_i with $\{(x, y, z) \mid (x, y) \in P_i, 0 \leq z \leq \phi_i\}$ creates a new edge in G; let $v_i v_j$ be such an edge. Then, clearly $\overline{P}_i \cap \overline{P}_j \neq \emptyset$. We will prove that v_j is a hovering vertex of v_i. Otherwise, $\phi_i < \phi_j$, $a_j > a_i$, or $b_i > b_j$. Suppose first that $\phi_i < \phi_j$. If $v_j \in V_U$, then v_i remains not connected to v_j after the replacement of P_i with $\{(x, y, z) \mid (x, y) \in P_i, 0 \leq z \leq \phi_i\}$, since $\phi_i < \phi_j$, which is a contradiction. If $v_j \in V_B$, then v_i is connected to v_j also before the replacement of P_i, since $\phi_i < \phi_j$ and $\overline{P}_i \cap \overline{P}_j \neq \emptyset$, which is again a contradiction. Thus, $\phi_i > \phi_j$. Suppose now that $a_j > a_i$ or $b_i > b_j$. Then, since $\phi_i > \phi_j$, we obtain for both cases that $\overline{P}_i \cap \overline{P}_j = \emptyset$, which is a contradiction. Thus, $a_j < a_i$, $b_i < b_j$, and $\phi_i > \phi_j$, i.e. v_j is a hovering vertex of v_i by Definition 5. ■

Definition 6 *A parallelepiped representation of a tolerance graph G is called* canonical *if every unbounded vertex is inevitable.*

For example, in the tolerance graph depicted in Figure 2, v_4 and v_8 are inevitable unbounded vertices, v_3 and v_6 are hovering vertices of v_4 and v_8, respectively, while v_2 and v_6 are evitable unbounded vertices. Therefore, this representation is not canonical for the graph G. However, if we replace P_i with $\{(x, y, z) \mid (x, y) \in P_i, 0 \leq z \leq \phi_i\}$ for $i = 2, 6$, we get a canonical representation for G.

In the following, we present an algorithm that constructs a canonical representation of a given tolerance graph G.

Definition 7 *Let $\alpha = (x_\alpha, y_\alpha)$ and $\beta = (x_\beta, y_\beta)$ be two points in the plane. Then α* dominates *β if $x_\alpha > x_\beta$ and $y_\alpha > y_\beta$. Given a set A of points, the point $\gamma \in A$ is called an* extreme point *of A if there is no point $\delta \in A$ that dominates γ. $Ex(A)$ is the set of the extreme points of A.*

Given a tolerance graph $G = (V, E)$ with the set $V = \{v_1, v_2, \ldots, v_n\}$ of vertices (and its parallelepiped representation), we can assume w.l.o.g. that $a_i < a_j$ whenever $i < j$. Recall that with every vertex v_i we associated the point $p_i = (x_i, y_i)$, where $x_i = b_i$ and $y_i = \frac{\pi}{2} - \phi_i$, respectively. We define for every $i = 1, 2, \ldots, n$ the set $A_i = \{p_1, p_2, \ldots, p_i\}$ of the points associated with the first i vertices of G.

Lemma 3 *Let $v_i \in V_U$ be an unbounded vertex of a tolerance graph G. Then:*

(a) If $p_i \in Ex(A_i)$ then v_i is evitable.

(b) If $p_i \notin Ex(A_i)$ and point p_j dominates p_i for some bounded vertex $v_j \in V_B$ with $j < i$ then v_i is inevitable and v_j is a hovering vertex of v_i.

Proof. (a) Assume, to the contrary, that v_i is inevitable. By Lemma 2 there is a hovering vertex v_j of v_i. But then, $x_i = b_i < b_j = x_j$ and $y_i = \frac{\pi}{2} - \phi_i < \frac{\pi}{2} - \phi_j = y_j$, while $a_j < a_i$, i.e. $j < i$. Therefore $p_j \in A_i$ and p_j dominates p_i, which is a contradiction, since $p_i \in Ex(A_i)$.

(b) Suppose that p_j dominates p_i, for some vertex $v_j \in V_B$ with $j < i$. The ordering of the vertices implies $a_j < a_i$, while $x_i < x_j$ and $y_i < y_j$ imply $b_i < b_j$ and $\phi_i > \phi_j$. Thus v_i is inevitable and v_j is a hovering vertex of v_i. ■

The following theorem shows that, given a parallelepiped representation of a tolerance graph G, we can construct in $\mathcal{O}(n \log n)$ a canonical representation of G. This result is crucial for the time complexity analysis of the algorithms of Section 3.2.

Theorem 2 *Every parallelepiped representation of a tolerance graph G with n vertices can be transformed by Algorithm 1 to a canonical representation of G in $\mathcal{O}(n \log n)$ time.*

Proof. We describe and analyze Algorithm 1 that generates a canonical representation of G. First, we sort the vertices $v_1, v_2, \ldots, v_n$ of G such that $a_i < a_j$ whenever $i < j$. Then, we process sequentially all vertices v_i of G. The bounded and the inevitable unbounded vertices will not be changed, while the evitable unbounded vertices will be replaced with bounded ones. At step i we update the set $Ex(A_i)$ of the extreme points of A_i (note that the set A_i remains unchanged during the algorithm). For two points p_{i_1}, p_{i_2} of $Ex(A_i)$, $x_{i_1} > x_{i_2}$ if and only if $y_{i_1} < y_{i_2}$. We store the elements of $Ex(A_i)$ in a list P, in which the points p_j are sorted increasingly according to their x values (or, equivalently, decreasingly according to their y values). Due to Lemma 3(a), and since during the algorithm the evitable unbounded vertices of G are replaced with bounded ones, after the process of vertex v_i, all points in the list P correspond to bounded vertices of G in the current parallelepiped representation.

We distinguish now the following cases:

Case 1. v_i is bounded. If there exists a point of P that dominates p_i then $p_i \notin Ex(A_i)$. Thus, we do not change P, and we continue to the process of v_{i+1}. If no point of P dominates p_i then $p_i \in Ex(A_i)$. Thus, we add p_i to P and we remove from P all points that are dominated by p_i.

Algorithm 1 Construction of a canonical representation of a tolerance graph G

Input: A parallelepiped representation R of a given tolerance graph G with n vertices
Output: A canonical representation R' of G

```
Sort the vertices of G, such that a_i < a_j whenever i < j
ℓ_0 ← min{x_i : 1 ≤ i ≤ n}; r_0 ← max{x_i : 1 ≤ i ≤ n}
p_s ← (ℓ_0 − 1, π/2); p_t ← (r_0 + 1, 0)
P ← (p_s, p_t); R' ← R
for i = 1 to n do
  Find the point p_j having the smallest x_j with x_j > x_i
  if y_j < y_i then {no point of P dominates p_i}
    Find the point p_k having the greatest x_k with x_k < x_i
    Find the point p_ℓ having the greatest y_ℓ with y_ℓ < y_i
    if x_k ≥ x_ℓ then
      Replace points p_ℓ, p_{ℓ+1} ..., p_k with point p_i in the list P
    else
      Insert point p_i between points p_k and p_ℓ in the list P
    if v_i ∈ V_U then {v_i is an evitable unbounded vertex}
      Replace P_i with {(x, y, z) | (x, y) ∈ P_i, 0 ≤ z ≤ φ_i} in R'
  else {y_j > y_i; p_j dominates p_i}
    if v_i ∈ V_U then {v_i is an inevitable unbounded vertex}
      v_j is a hovering vertex of v_i
return R'
```

Case 2. v_i is unbounded. If there exists a point $p_j \in P$ that dominates p_i then $p_i \notin Ex(A_i)$, while Lemma 3(b) implies that v_i is inevitable and v_j is a hovering vertex of v_i. Thus, similarly to Case 1, we do not change P, and we continue to the process of v_{i+1}. If no point of P dominates p_i then $p_i \in Ex(A_i)$. Thus, we add the point p_i to P and remove from P all points that are dominated by p_i. In this case, v_i is evitable by Lemma 3(a). Hence, we replace P_i with $\{(x, y, z) \mid (x, y) \in P_i, 0 \leq z \leq \phi_i\}$ in the current parallelepiped representation of G and we consider from now on v_i as a bounded vertex.

It follows that after the process of each vertex v_i (either bounded or unbounded) the list P stores the points of $Ex(A_i)$. Furthermore, at every iteration of the algorithm, all points of the list P correspond to bounded vertices in the current parallelepiped representation of G.

The processing of vertex v_i is done by executing three binary searches in the list P as follows. Let $\ell_0 = \min\{x_i \mid 1 \leq i \leq n\}$ and $r_0 = \max\{x_i \mid 1 \leq i \leq n\}$. For convenience, we add two dummy points $p_s = (\ell_0 - 1, \frac{\pi}{2})$ and $p_t = (r_0 + 1, 0)$. First, we find the point $p_j \in P$ with the smallest value x_j, such that $x_j > x_i$ (see Figure 3). Note that $p_i \in Ex(A_i)$ if and only if $y_j < y_i$. If $y_j > y_i$ then p_j dominates p_i (see Figure 3(a)). Thus, if $v_i \in V_U$, Lemma 3(b) implies that v_i is an inevitable unbounded vertex and v_j is a hovering vertex of v_i. In the opposite case $y_j < y_i$, we have to add p_i to P. In order to remove from P all points that are dominated by p_i, we execute binary search two more times. In particular, we find the points p_k and p_ℓ of P with the greatest values x_k and y_ℓ, respectively, such that $x_k < x_i$ and $y_\ell < y_i$ (see Figure 3(b)). If there are some points of P that are dominated by p_i, then p_k and p_ℓ have the greatest and smallest values x_k and x_ℓ among them, respectively, and $x_k \geq x_\ell$. In this case, we replace all points $p_\ell, p_{\ell+1}, \ldots, p_k$ with the point p_i in the list P. Otherwise, if no point of P is dominated by p_i, then $x_k < x_\ell$. In this case, we remove no point from P and we

insert p_i between p_k and p_ℓ in P.

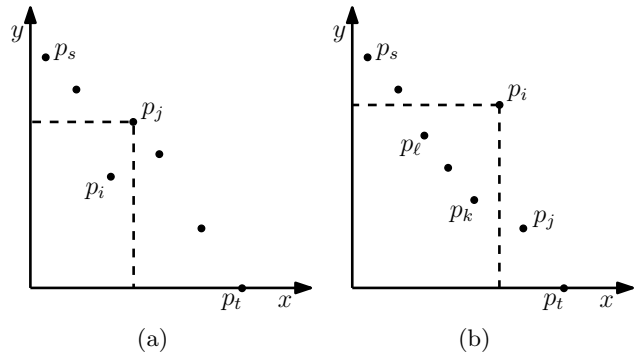

Figure 3: The cases where the associated point p_i to the currently processed vertex v_i is (a) dominated by the point p_j in A_i and (b) an extreme point of the set A_i.

Finally, after processing all vertices v_i of G, we return a canonical representation of the given tolerance graph G, in which every vertex that remains unbounded has a hovering vertex assigned to it. Since the processing of every vertex can be done in $\mathcal{O}(\log n)$ time by executing three binary searches, and since the sorting of the vertices can be done in $\mathcal{O}(n \log n)$ time, the running time of Algorithm 1 is $\mathcal{O}(n \log n)$. ■

3.2 Minimum coloring

In the next theorem we present an optimal $\mathcal{O}(n \log n)$ algorithm for computing a minimum coloring of a tolerance graph G with n vertices, given a parallelepiped representation of G. The informal description of the algorithm is identical to the one in [12], which has running time $\mathcal{O}(n^2)$; the difference is in the fact that we use our new representation, in order to improve the time complexity.

Algorithm 2 Minimum coloring of a tolerance graph G

Input: A parallelepiped representation of a given tolerance graph G
Output: A minimum coloring of G

Construct a canonical representation of G by Algorithm 1, where a hovering vertex is associated with every inevitable unbounded vertex
Color $G[V_B]$ by the algorithm of [6]
for every inevitable unbounded vertex $v_i \in V_U$ **do**
  Assign to v_i the same color as its hovering vertex in $G[V_B]$

Theorem 3 *A minimum coloring of a tolerance graph G with n vertices can be computed in $\mathcal{O}(n \log n)$ time.*

Proof. We present Algorithm 2 that computes a minimum coloring of G. Given a parallelepiped representation of G, we construct a canonical representation of G in $\mathcal{O}(n \log n)$ time by Algorithm 1. V_B and V_U are the sets of bounded and inevitable unbounded vertices of G in the latter representation, respectively. In particular, Algorithm 1 associates a hovering vertex $v_j \in V_B$ with every inevitable unbounded vertex $v_i \in V_U$. We find a minimum proper coloring of the bounded tolerance graph $G[V_B]$ in $\mathcal{O}(n \log n)$ time using the algorithm of [6]. Finally, we associate with every inevitable unbounded vertex $v_i \in V_U$ the same color as that of its hovering vertex $v_j \in V_B$ in the coloring of $G[V_B]$.

Consider an arbitrary inevitable unbounded vertex $v_i \in V_U$ and its hovering vertex $v_j \in V_B$. Following Definition 5, $\overline{P}_i \cap \overline{P}_j \neq \emptyset$ and $\phi_i > \phi_j$. Consider a vertex v_k of G, such that $v_i v_k \in E$. It

follows that $v_k \in V_B$, since no two unbounded vertices are adjacent in G. Furthermore, since $v_i v_k \in E$, it follows that $\overline{P}_i \cap \overline{P}_k \neq \emptyset$ and $\phi_k > \phi_i$. Then $\overline{P}_j \cap \overline{P}_k \neq \emptyset$, and thus $P_j \cap P_k \neq \emptyset$, i.e. $v_j v_k \in E$, since both v_j and v_k are bounded vertices. It follows that v_k does not have the same color as v_j in the proper coloring of $G[V_B]$, and thus the resulting coloring of G is proper. Finally, since both colorings of $G[V_B]$ and of G have the same number of colors, it follows that this proper coloring of G is minimum. Since the coloring of $G[V_B]$ can be done in $\mathcal{O}(n \log n)$ time and the coloring of all inevitable unbounded vertices $v_i \in V_U$ can be done in $\mathcal{O}(n)$ time, Algorithm 2 returns a minimum proper coloring G in $\mathcal{O}(n \log n)$ time. ■

3.3 Maximum clique

In the next theorem we prove that a maximum clique of a tolerance graph G with n vertices can be computed in optimal $\mathcal{O}(n \log n)$ time, given a parallelepiped representation of G. This theorem follows from Theorem 2 and from the clique algorithm presented in [6], and it improves the best known $\mathcal{O}(n^2)$ running time mentioned in [12].

Theorem 4 *A maximum clique of a tolerance graph G with n vertices can be computed in $\mathcal{O}(n \log n)$ time.*

Proof. We compute first a canonical representation of G in $\mathcal{O}(n \log n)$ time by Algorithm 1. The proof of Theorem 3 implies that $\chi(G) = \chi(G[V_B])$, where $\chi(H)$ denotes the chromaric number of a given graph H. Since tolerance graphs are perfect graphs [11], $\omega(G) = \chi(G)$ and $\omega(G[V_B]) = \chi(G[V_B])$, where $\omega(H)$ denotes the clique number of a given graph H. It follows that $\omega(G) = \omega(G[V_B])$. We compute now a maximum clique Q of the bounded tolerance graph $G[V_B]$ in $\mathcal{O}(n \log n)$ time. This can be done by the algorithm presented in [6] that computes a maximum clique in a trapezoid graph, since bounded tolerance graphs are trapezoid graphs [13]. Since $\omega(G) = \omega(G[V_B])$, Q is a maximum clique of G as well. ■

3.4 Optimality of the running time

In this section we use permutation graphs [13]. Given a sequence $S = a_1, a_2, \ldots, a_n$ of numbers, a *subsequence* of S is a sequence $S' = a_{i_1}, a_{i_2}, \ldots, a_{i_k}$, where $a_{i_j} \in S$ for every $j \in \{1, 2, \ldots, k\}$, and $1 \leq i_1 < i_2 < \ldots < i_k \leq n$. S' is called an *increasing subsequence* of S, if $a_{i_1} < a_{i_2} < \ldots < a_{i_k}$. Clearly, increasing subsequences in a permutation graph G correspond to independent sets of G, while increasing subsequences in the complement $\overline{G}$ of G correspond to cliques of G, where $\overline{G}$ is also a permutation graph. Since $\Omega(n \log n)$ is a lower time bound for computing the length of a longest increasing subsequence in a permutation [6, 8], the same lower time bound holds for computing a maximum clique and a maximum independent set in a permutation graph G. Furthermore, since permutation graphs are perfect graphs [9], the chromatic number $\chi(G)$ of a permutation graph G equals the clique number $\omega(G)$ of G. Thus, $\Omega(n \log n)$ is a lower time bound for computing the chromatic number of a permutation graph. Finally, since the class of permutation graphs is a subclass of tolerance graphs [13], the same lower bounds hold for tolerance graphs. It follows that the algorithms in Theorems 3 and 4 for computing a minimum coloring and a maximum clique in tolerance graphs are optimal.

4 Weighted Independent Set Algorithm in $\mathcal{O}(n^2)$

In this section we present an algorithm for computing a maximum weight independent set in a tolerance graph $G = (V, E)$ with n vertices in $\mathcal{O}(n^2)$ time, given a parallelepiped representation of G, and a weight $w(v_i) > 0$ for every vertex v_i of G. The proposed algorithm improves the running time

$\mathcal{O}(n^3)$ of the one presented in [13]. In the following, consider as above the partition of the vertex set V into the sets V_B and V_U of bounded and unbounded vertices of G, respectively.

Similarly to [13], we add two isolated bounded vertices v_s and v_t to G with weights $w(v_s) = w(v_t) = 0$, such that the corresponding parallelepipeds P_s and P_t lie completely to the left and to the right of all other parallelepipeds of G, respectively. Since both v_s and v_t are bounded vertices, we augment the set V_B by the vertices v_s and v_t. In particular, we define the set of vertices $V'_B = V_B \cup \{v_s, v_t\}$ and the tolerance graph $G' = (V', E)$, where $V' = V'_B \cup V_U$. Since $G'[V'_B]$ is a bounded tolerance graph, it is a co-comparability graph as well [11, 13]. A transitive orientation of the comparability graph $\overline{G'[V'_B]}$ can be obtained by directing each edge according to the upper left endpoints of the parallelograms $\overline{P}_i$. Formally, let $(V'_B, \prec)$ be the partial order defined on the bounded vertices V'_B, such that $v_i \prec v_j$ if and only if $v_i v_j \notin E$ and $c_i < c_j$. Recall that a *chain* of elements in a partial order is a set of mutually comparable elements in this order [4].

Observation 2 ([13]) *The independent sets of $G[V_B]$ are in one-to-one correspondence with the chains in the partial order $(V'_B, \prec)$ from v_s to v_t.*

For the sequel, recall that for every unbounded vertex $v_k \in V_U$ the parallelepiped P_k degenerates to a line segment, while the upper endpoints b_k and c_k of the parallelogram $\overline{P}_k$ coincide, i.e. $b_k = c_k$.

Definition 8 *For every $v_i, v_j \in V'_B$ with $v_i \prec v_j$, $L_i(j) = \{v_k \in V_U \mid b_i < c_k < c_j, v_i v_k \notin E\}$ and its weight $w(L_i(j)) = \sum_{v \in L_i(j)} w(v)$.*

Definition 9 *For every $v_j \in V'_B$, $R_j = \{v_k \in V_U \mid c_j < c_k < b_j, v_j v_k \notin E\}$ and its weight $w(R_j) = \sum_{v \in R_j} w(v)$.*

For every pair of bounded vertices $v_i, v_j \in V'_B$ with $v_i \prec v_j$, the set $L_i(j)$ consists of those unbounded vertices $v_k \in V_U$, for which $v_i v_k \notin E$ and whose upper endpoint $b_k = c_k$ of $\overline{P}_k$ lies between $\overline{P}_i$ and $\overline{P}_j$. Furthermore, $v_j v_k \notin E$ for every vertex $v_k \in L_i(j)$. Indeed, in the case where $\overline{P}_k \cap \overline{P}_j \neq \emptyset$, it holds $\phi_k > \phi_j$, since $b_k = c_k < c_j$, and thus $P_k \cap P_j = \emptyset$. Similarly, the set R_j consists of those unbounded vertices $v_k \in V_U$, for which $v_j v_k \notin E$ and whose upper endpoint $b_k = c_k$ of $\overline{P}_k$ lies between the upper endpoints c_j and b_j of $\overline{P}_j$. Furthermore, $v_i v_k \notin E$ for every vertex $v_k \in R_j$ as well. Indeed, since $v_j v_k \notin E$, it follows that $\phi_k > \phi_j$, and thus, $\overline{P}_i \cap \overline{P}_k = \emptyset$ and $P_i \cap P_k = \emptyset$. In particular, in the example of Figure 4, $L_1(2) = \{v_3, v_5\}$ and $R_2 = \{v_6\}$. In this figure, the line segments that correspond to the unbounded vertices v_4 and v_7, respectively, are drawn with dotted lines to illustrate the fact that $v_4 v_1 \in E$ and $v_7 v_2 \in E$.

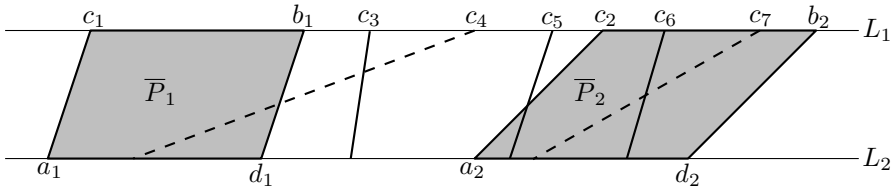

Figure 4: The parallelograms $\overline{P}_i$, $i = 1, 2, \ldots, 7$ of a tolerance graph with the sets $V_B = \{v_1, v_2\}$ and $V_U = \{v_3, v_4, \ldots, v_7\}$ of bounded and unbounded vertices, respectively. In this graph, $L_1(2) = \{v_3, v_5\}$, $R_2 = \{v_6\}$ and $S(v_1, v_2) = \{v_3, v_5, v_6\}$.

Definition 10 ([13]) *For every $v_i, v_j \in V'_B$ with $v_i \prec v_j$, $S(v_i, v_j) = \{v_k \in V_U \mid v_i v_k, v_j v_k \notin E, b_i < c_k < b_j\}$.*

Algorithm 3 Maximum weight independent set of a tolerance graph G

Input: A parallelepiped representation of a given tolerance graph G
Output: The value of a maximum weight independent set of G

Add the dummy bounded vertices v_s, v_t to G, such that P_s and P_t lie completely to the left and to the right of all other parallelepipeds of G, respectively
$V'_B \leftarrow V_B \cup \{v_s, v_t\}$
Construct the partial ordering $(V'_B, \prec)$ of the bounded vertices V'_B
Sort the bounded vertices V'_B, such that $c_i < c_j$ whenever $i < j$

for $j = 1$ to $|V'_B|$ **do**
  $W(v_j) \leftarrow 0$
  Compute the value $w(R_j)$

for $i = 1$ to $|V'_B|$ **do**
  for every $v_j \in V'_B$ with $v_i \prec v_j$ **do**
    Update the value $w(L_i(j))$
    if $W(v_j) < (w(v_j) + w(R_j)) + W(v_i) + w(L_i(j))$ **then**
      $W(v_j) \leftarrow (w(v_j) + w(R_j)) + W(v_i) + w(L_i(j))$
return $W(v_t)$

Observation 3 *For every pair of bounded vertices $v_i, v_j \in V'_B$ with $v_i \prec v_j$,*

$$S(v_i, v_j) = L_i(j) \cup R_j \tag{1}$$

Furthermore, $L_i(j) \subseteq L_i(\ell)$ for every triple $\{v_i, v_j, v_\ell\}$ of bounded vertices, where $v_i \prec v_j$, $v_i \prec v_\ell$ and $c_j < c_\ell$.

In particular, in the example of Figure 4, $S(v_1, v_2) = L_1(2) \cup R_2 = \{v_3, v_5, v_6\}$.

Lemma 4 ([13]) *Given a tolerance graph G with a set of positive weights for the vertices of G, any maximum weight independent set of G consists of a chain of bounded vertices $v_{x_1} \prec v_{x_2} \prec \ldots \prec v_{x_k}$ together with the union of the sets $\cup\{S(v_{x_i}, v_{x_{i+1}}) \mid i = 0, 1, \ldots, k\}$, where $v_{x_0} = v_s$ and $v_{x_{k+1}} = v_t$.*

Now, using Lemma 4 and Observation 3, we can present Algorithm 3, which improves the running time $\mathcal{O}(n^3)$ of the one presented in [13].

Theorem 5 *A maximum weight independent set of a tolerance graph G with n vertices can be computed using Algorithm 3 in $\mathcal{O}(n^2)$ time.*

Proof. We present Algorithm 3 that computes the value of a maximum weight independent set of G. A slight modification of Algorithm 3 returns a maximum weight independent set of G, instead of its value. First, we construct the partial order $(V'_B, \prec)$ defined on the bounded vertices $V'_B = V_B \cup \{v_s, v_t\}$, such that $v_i \prec v_j$ whenever $v_i v_j \notin E$ and $c_i < c_j$. This can be done in $\mathcal{O}(n^2)$ time. Then, we sort the bounded vertices of V'_B, such that $c_i < c_j$ whenever $i < j$. This can be done in $\mathcal{O}(n \log n)$ time. As a preprocessing step, we compute for every bounded vertex $v_j \in V'_B$ the set R_j and its weight $w(R_j)$ in linear $\mathcal{O}(n)$ time by visiting at most all unbounded vertices $v_k \in V_U$. Thus, all values $w(R_j)$ are computed in $\mathcal{O}(n^2)$ time.

We associate with each bounded vertex $v_j \in V'_B$ a cumulative weight $W(v_j)$ defined as follows:

$$\begin{aligned} W(v_s) &= 0 \\ W(v_j) &= (w(v_j) + w(R_j)) + \max_{v_i \prec v_j}\{W(v_i) + w(L_i(j))\}, \text{ for every } v_j \in V'_B \setminus \{v_s\} \end{aligned} \tag{2}$$

The cumulative weight $W(v_j)$ of an arbitrary bounded vertex $v_j \in V'_B$ equals the maximum weight of an independent set S of vertices v_k (both bounded and unbounded), for which $b_k \leq b_j$ and $v_j \in S$. Initially all values $W(v_j)$ are set to zero.

In the main part of Algorithm 3, we process sequentially all bounded vertices $v_i \in V'_B$. For every such vertex v_i, we update sequentially the cumulative weights $W(v_j)$ for all bounded vertices $v_j \in V'_B$ with $v_i \prec v_j$ by comparing the current value of $W(v_j)$ with the value $(w(v_j) + w(R_j)) + W(v_i) + w(L_i(j))$, and by storing the greatest of them in $W(v_j)$. After all bounded vertices of V'_B have been processed, the value of the maximum weight independent set of G is stored in $W(v_t)$, due to Lemma 4 and Observation 3.

While processing the bounded vertex v_i, we compute the values $w(L_i(j))$ sequentially for every j, where $v_i \prec v_j$, as follows. Let v_{j_1}, v_{j_2} be two bounded vertices that are visited consecutively by the algorithm, during the process of vertex v_i. Then, due to Observation 3, we compute the value $w(L_i(j_2))$ by adding to the previous value $w(L_i(j_1))$ the weights of all unbounded vertices $v_k \in V_U$, whose upper endpoints c_k lie between c_{j_1} and c_{j_2}.

Since we visit all bounded and all unbounded vertices of the graph at most once during the process of v_i, this can be done in $\mathcal{O}(n)$ time. Thus, since there are in total at most $n + 2$ bounded vertices $v_i \in V'_B$, Algorithm 3 returns the value of the maximum weight independent set of G in $\mathcal{O}(n^2)$ time. Finally, observe that, storing at every step of Algorithm 3 the independent sets that correspond to the values $W(v_i)$, and removing at the end the vertices v_s and v_t, the algorithm returns at the same time a maximum weight independent set of G, instead of its value. ■

5 Conclusions and Further Research

In this article we proposed the first non-trivial intersection model for general tolerance graphs, given by parallelepipeds in the three-dimensional space. This representation generalizes the parallelogram representation of bounded tolerance graphs. Using this representation, we presented improved algorithms for computing a minimum coloring, a maximum clique, and a maximum weight independent set on a tolerance graph. The running times of the first two algorithms are optimal. It can be expected that this representation will prove useful in improving the running time of other algorithms for the class of tolerance graphs.

As mentioned in Section 1, the complexity of the recognition problem for tolerance and bounded tolerance graphs is possibly the main open problem in this class of graphs. Even when the input graph is known to be a tolerance graph, it is not known how to obtain a tolerance representation for it [22]. Moreover, given a tolerance graph, it is not known how to decide in polynomial time whether it is a bounded tolerance graph [22].

References

[1] K. P. Bogart, P. C. Fishburn, G. Isaak, and L. Langley. Proper and unit tolerance graphs. *Discrete Applied Mathematics*, 60(1-3):99–117, 1995.

[2] A. H. Busch. A characterization of triangle-free tolerance graphs. *Discrete Applied Mathematics*, 154(3):471–477, 2006.

[3] A. H. Busch and G. Isaak. Recognizing bipartite tolerance graphs in linear time. In *33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 12–20, 2007.

[4] R. Diestel. *Graph Theory*. Springer-Verlag, Berlin, 3rd edition, 2005.

[5] S. Felsner. Tolerance graphs and orders. *Journal of Graph Theory*, 28:129–140, 1998.

[6] S. Felsner, R. Müller, and L. Wernisch. Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, 74:13–32, 1997.

[7] P. Fishburn and W. Trotter. Split semiorders. *Discrete Mathematics*, 195:111–126, 1999.

[8] M. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11:29–35, 1975.

[9] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol. 57)*. North-Holland Publishing Co., 2004.

[10] M. Golumbic and C. Monma. A generalization of interval graphs with tolerances. In *Proceedings of the 13th Southeastern Conference on Combinatorics, Graph Theory and Computing, Congressus Numerantium 35*, pages 321–331, 1982.

[11] M. Golumbic, C. Monma, and W. Trotter. Tolerance graphs. *Discrete Applied Mathematics*, 9(2):157–170, 1984.

[12] M. Golumbic and A. Siani. Coloring algorithms for tolerance graphs: Reasoning and scheduling with interval constraints. In *Joint International Conferences on Artificial Intelligence, Automated Reasoning, and Symbolic Computation (AISC/Calculemus)*, pages 196–207, 2002.

[13] M. Golumbic and A. Trenk. *Tolerance Graphs*. Cambridge Studies in Advanced Mathematics, 2004.

[14] M. Grötshcel, L. Lovász, and A. Schrijver. The Ellipsoid Method and its Consequences in Combinatorial Optimization. *Combinatorica*, 1:169–197, 1981.

[15] R. B. Hayward and R. Shamir. A note on tolerance graph recognition. *Discrete Applied Mathematics*, 143(1-3):307–311, 2004.

[16] G. Isaak, K. Nyman, and A. Trenk. A hierarchy of classes of bounded bitolerance orders. *Ars Combinatoria*, 69, 2003.

[17] M. Jacobson and F. McMorris. Sum-tolerance proper interval graphs are precisely sum-tolerance unit interval graphs. *Journal of Combinatorics, Information and System Science*, 16:25–28, 1991.

[18] M. Kaufmann, J. Kratochvíl, K. A. Lehmann, and A. R. Subramanian. Max-tolerance graphs as intersection graphs: cliques, cycles, and recognition. In *17th annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 832–841, 2006.

[19] J. M. Keil and P. Belleville. Dominating the complements of bounded tolerance graphs and the complements of trapezoid graphs. *Discrete Applied Mathematics*, 140(1-3):73–89, 2004.

[20] L. Langley. *Interval tolerance orders and dimension*. PhD thesis, Dartmouth College, June 1993.

[21] T. McKee and F. McMorris. *Topics in Intersection Graph Theory*. Society for Industrial and Applied Mathematics (SIAM), 1999.

[22] G. Narasimhan and R. Manber. Stability and chromatic number of tolerance graphs. *Discrete Applied Mathematics*, 36:47–56, 1992.

[23] P. Zhang, E. A. Schon, S. G. Fischer, E. Cayanis, J. Weiss, S. Kistler, and P. E. Bourne. An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA. *CABIOS*, 10:309–317, 1994.

The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete

George B. Mertzios* Ignasi Sau† Shmuel Zaks‡

Abstract

Tolerance graphs model interval relations in such a way that intervals can tolerate a certain degree of overlap without being in conflict. This subclass of perfect graphs has been extensively studied, due to both its interesting structure and its numerous applications (in bioinformatics, constrained-based temporal reasoning, resource allocation, and scheduling problems, among others). Several efficient algorithms for optimization problems that are NP-hard in general graphs have been designed for tolerance graphs. In spite of this, the recognition of tolerance graphs – namely, the problem of deciding whether a given graph is a tolerance graph – as well as the recognition of their main subclass of bounded tolerance graphs, have been the most fundamental open problems on this class of graphs (cf. the book on tolerance graphs [14]) since their introduction in 1982 [11]. In this article we prove that both recognition problems are NP-complete, even in the case where the input graph is a trapezoid graph. The presented results are surprising because, on the one hand, most subclasses of perfect graphs admit polynomial recognition algorithms and, on the other hand, bounded tolerance graphs were believed to be efficiently recognizable as they are a natural special case of trapezoid graphs, which can be recognized in polynomial time. For our reduction we extend the notion of an acyclic orientation of permutation and trapezoid graphs. Our main tool is a new algorithm that transforms a given trapezoid graph into a permutation graph, while preserving this new acyclic orientation property.

Keywords: Tolerance graphs, bounded tolerance graphs, recognition, NP-complete, trapezoid graphs, permutation graphs.

*Department of Computer Science, RWTH Aachen University, Germany. Email: mertzios@cs.rwth-aachen.de

†Mascotte joint Project of INRIA/CNRS/UNSA, Sophia-Antipolis, France; and Graph Theory and Combinatorics Group, Applied Maths. IV Dept. of UPC, Barcelona, Spain. Email: ignasi.sau@sophia.inria.fr

‡Department of Computer Science, Technion, Haifa, Israel. Email: zaks@cs.technion.ac.il

1 Introduction

1.1 Tolerance graphs and related graph classes

A simple undirected graph $G = (V, E)$ on n vertices is a *tolerance* graph if there exists a collection $I = \{I_i \mid i = 1, 2, \ldots, n\}$ of closed intervals on the real line and a set $t = \{t_i \mid i = 1, 2, \ldots, n\}$ of positive numbers, such that for any two vertices $v_i, v_j \in V$, $v_i v_j \in E$ if and only if $|I_i \cap I_j| \geq \min\{t_i, t_j\}$. The pair $\langle I, t\rangle$ is called a *tolerance representation* of G. If G has a tolerance representation $\langle I, t\rangle$, such that $t_i \leq |I_i|$ for every $i = 1, 2, \ldots, n$, then G is called a *bounded tolerance* graph and $\langle I, t\rangle$ a *bounded tolerance representation* of G.

Tolerance graphs were introduced in [11], in order to generalize some of the well known applications of interval graphs. The main motivation was in the context of resource allocation and scheduling problems, in which resources, such as rooms and vehicles, can tolerate sharing among users [14]. If we replace in the definition of tolerance graphs the operator *min* by the operator *max*, we obtain the class of *max-tolerance* graphs. Both tolerance and max-tolerance graphs find in a natural way applications in biology and bioinformatics, as in the comparison of DNA sequences from different organisms or individuals [19], by making use of a software tool like BLAST [1]. Tolerance graphs find numerous other applications in constrained-based temporal reasoning, data transmission through networks to efficiently scheduling aircraft and crews, as well as contributing to genetic analysis and studies of the brain [13,14]. This class of graphs has attracted many research efforts [2, 4, 8, 12–14, 17, 20, 24, 26], as it generalizes in a natural way both interval graphs (when all tolerances are equal) and permutation graphs (when $t_i = |I_i|$ for every $i = 1, 2, \ldots, n$) [11]. For a detailed survey on tolerance graphs we refer to [14].

A graph is *perfect* if the chromatic number of every induced subgraph equals the clique number of that subgraph. Several difficult combinatorial problems can be solved efficiently, i.e. in polynomial time, on the class of perfect graphs, such as minimum coloring, maximum clique, and independent set [16]. Thus, since the class of tolerance graphs is a subclass of perfect graphs [12], there exist polynomial algorithms for these problems on tolerance and bounded tolerance graphs as well. In spite of this, faster algorithms have been designed for tolerance and bounded tolerance graphs, which exploit their special structure [13, 14, 24, 26].

A *comparability* graph is a graph which can be transitively oriented. A *co-comparability* graph is a graph whose complement is a comparability graph. A *trapezoid* (resp. *parallelogram* and *permutation*) graph is the intersection graph of trapezoids (resp. parallelograms and line segments) between two parallel lines L_1 and L_2 [10]. Such a representation with trapezoids (resp. parallelograms and line segments) is called a *trapezoid* (resp. *parallelogram* and *permutation*) *representation* of this graph. A graph is bounded tolerance if and only if it is a parallelogram graph [2, 21]. Permutation graphs are a strict subset of parallelogram graphs [3]. Furthermore, parallelogram graphs are a strict subset of trapezoid graphs [28], and both are subsets of co-comparability graphs [10, 14]. On the contrary, tolerance graphs are not even co-comparability graphs [10, 14]. Recently, we have presented in [24] a natural intersection model for general tolerance graphs, given by parallelepipeds in the three-dimensional space. This representation generalizes the parallelogram representation of bounded tolerance graphs, and has been used to improve the time complexity of minimum coloring, maximum clique, and weighted independent set algorithms on tolerance graphs [24].

Although tolerance and bounded tolerance graphs have been studied extensively, the recognition problems for both these classes have been the most fundamental open problems since their introduction in 1982 [5, 10, 14]. Therefore, all existing algorithms assume that, along with the input tolerance graph, a tolerance representation of it is given. The only result about the complexity of recognizing tolerance and bounded tolerance graphs is that they have a (non-trivial) polynomial sized tolerance representation, hence the problems of recognizing tolerance and bounded tolerance graphs are in the class NP [17]. Recently, a linear time recognition algorithm for the subclass of *bipartite tolerance* graphs has been presented in [5]. Furthermore, the class of trapezoid graphs (which strictly contains parallelogram, i.e. bounded tolerance, graphs [28]) can be also recognized in polynomial time [6, 22, 30]. On the other hand, the recognition of max-tolerance graphs is known

to be NP-hard [19]. Unfortunately, the structure of max-tolerance graphs differs significantly from that of tolerance graphs (max-tolerance graphs are not even perfect, as they can contain induced C_5's [19]), so the technique used in [19] does not carry over to tolerance graphs.

Since very few subclasses of perfect graphs are known to be NP-hard to recognize (for instance, *perfectly orderable* graphs [25] or *EPT* graphs [15]), it was believed that the recognition of tolerance graphs was in P. Furthermore, as bounded tolerance graphs are equivalent to parallelogram graphs [2, 21], which constitute a natural subclass of trapezoid graphs and have a very similar structure, it was plausible that their recognition was also in P.

1.2 Our contribution

In this article, we establish the complexity of recognizing tolerance and bounded tolerance graphs. Namely, we prove that both problems are surprisingly NP-complete, by providing a reduction from the monotone-Not-All-Equal-3-SAT (monotone-NAE-3-SAT) problem. Consider a boolean formula ϕ in conjunctive normal form with three literals in every clause (3-CNF), which is monotone, i.e. no variable is negated. The formula ϕ is called NAE-satisfiable if there exists a truth assignment of the variables of ϕ, such that every clause has at least one true variable and one false variable. Given a monotone 3-CNF formula ϕ, we construct a trapezoid graph H_ϕ, which is parallelogram, i.e. bounded tolerance, if and only if ϕ is NAE-satisfiable. Moreover, we prove that the constructed graph H_ϕ is tolerance if and only if it is bounded tolerance. Thus, since the recognition of tolerance and of bounded tolerance graphs are in the class NP [17], it follows that these problems are both NP-complete. Actually, our results imply that the recognition problems remain NP-complete even if the given graph is trapezoid, since the constructed graph H_ϕ is trapezoid.

For our reduction we extend the notion of an acyclic orientation of permutation and trapezoid graphs. Our main tool is a new algorithm that transforms a given trapezoid graph into a permutation graph by splitting some specific vertices, while preserving this new acyclic orientation property. One of the main advantages of this algorithm is that the constructed permutation graph does not depend on any particular trapezoid representation of the input graph G. Moreover, this approach based on splitting vertices has already been proved useful for the design of polynomial recognition algorithms for other classes of graphs [23].

Organization of the paper. We first present in Section 2 several properties of permutation and trapezoid graphs, as well as the algorithm Split-U, which constructs a permutation graph from a trapezoid graph. In Section 3 we present the reduction of the monotone-NAE-3-SAT problem to the recognition of bounded tolerance graphs. In Section 4 we prove that this reduction can be extended to the recognition of general tolerance graphs. Finally, we discuss the presented results and further research directions in Section 5. Due to space limitations, some proofs are given in the Appendix.

2 Trapezoid graphs and representations

In this section we first introduce (in Section 2.1) the notion of an *acyclic representation* of permutation and of trapezoid graphs. This is followed (in Section 2.2) by some structural properties of trapezoid graphs, which will be used in the sequel for the splitting algorithm Split-U. Given a trapezoid graph G and a vertex subset U of G with certain properties, this algorithm constructs a permutation graph $G^{\#}(U)$ with $2|U|$ vertices, which is independent on any particular trapezoid representation of the input graph G.

Notation. We consider in this article simple undirected and directed graphs with no loops or multiple edges. In an undirected graph G, the edge between vertices u and v is denoted by uv, and in this case u and v are said to be *adjacent* in G. If the graph G is directed, we denote by uv the arc from u to v. Given a graph $G = (V, E)$ and a subset $S \subseteq V$, $G[S]$ denotes the induced subgraph of G on the vertices in S, and we use $E[S]$ to denote $E(G[S])$. Whenever we

deal with a trapezoid (resp. permutation and bounded tolerance, i.e. parallelogram) graph, we will consider w.l.o.g. a trapezoid (resp. permutation and parallelogram) representation, in which all endpoints of the trapezoids (resp. line segments and parallelograms) are distinct [9, 14, 18]. Given a permutation graph P along with a permutation representation R, we may not distinguish in the following between a vertex of P and the corresponding line segment in R, whenever it is clear from the context. Furthermore, with a slight abuse of notation, we will refer to the line segments of a permutation representation just as *lines*.

2.1 Acyclic permutation and trapezoid representations

Let $P = (V, E)$ be a permutation graph and R be a permutation representation of P. For a vertex $u \in V$, denote by $\theta_R(u)$ the angle of the line of u with L_2 in R. The class of permutation graphs is the intersection of comparability and co-comparability graphs [10]. Thus, given a permutation representation R of P, we can define two partial orders $(V, <_R)$ and $(V, \ll_R)$ on the vertices of P [10]. Namely, for two vertices u and v of G, $u <_R v$ if and only if $uv \in E$ and $\theta_R(u) < \theta_R(v)$, while $u \ll_R v$ if and only if $uv \notin E$ and u lies to the left of v in R. The partial order $(V, <_R)$ implies a transitive orientation Φ_R of P, such that $uv \in \Phi_R$ whenever $u <_R v$.

Let $G = (V, E)$ be a trapezoid graph, and R be a trapezoid representation of G, where for any vertex $u \in V$, the trapezoid corresponding to u in R is denoted by T_u. Since trapezoid graphs are also co-comparability graphs [10], we can similarly define the partial order $(V, \ll_R)$ on the vertices of G, such that $u \ll_R v$ if and only if $uv \notin E$ and T_u lies completely to the left of T_v in R. In this case, we may denote also $T_u \ll_R T_v$, instead of $u \ll_R v$.

In a given trapezoid representation R of a trapezoid graph G, we denote by $l(T_u)$ and $r(T_u)$ the left and the right line of T_u in R, respectively. Similarly to the case of permutation graphs, we use the relation $\ll_R$ for the lines $l(T_u)$ and $r(T_u)$, e.g. $l(T_u) \ll_R r(T_v)$ means that the line $l(T_u)$ lies to the left of the line $r(T_v)$ in R. Moreover, if the trapezoids of all vertices of a subset $S \subseteq V$ lie completely to the left (resp. right) of the trapezoid T_u in R, we write $R(S) \ll_R T_u$ (resp. $T_u \ll_R R(S)$). Note that there are several trapezoid representations of a particular trapezoid graph G. Given one such representation R, we can obtain another one R' by *vertical axis flipping* of R, i.e. R' is the mirror image of R along an imaginary line perpendicular to L_1 and L_2. Moreover, we can obtain another representation R'' of G by *horizontal axis flipping* of R, i.e. R'' is the mirror image of R along an imaginary line parallel to L_1 and L_2. We will use extensively these two basic operations throughout the article.

Definition 1 *Let P be a permutation graph with $2n$ vertices $\{u_1^1, u_1^2, u_2^1, u_2^2, \ldots, u_n^1, u_n^2\}$. Let R be a permutation representation and Φ_R be the corresponding transitive orientation of P. The simple directed graph F_R is obtained by merging u_i^1 and u_i^2 into a single vertex u_i, for every $i = 1, 2, \ldots, n$, where the arc directions of F_R are implied by the corresponding directions in Φ_R. Then,*

1. *R is an* acyclic permutation representation with respect to $\{u_i^1, u_i^2\}_{i=1}^n$*[*], if F_R has no directed cycle,*
2. *P is an* acyclic permutation graph with respect to $\{u_i^1, u_i^2\}_{i=1}^n$*, if P has an acyclic representation R with respect to $\{u_i^1, u_i^2\}_{i=1}^n$.*

In Figure 1 we show an example of a permutation graph P with six vertices in Figure 1(a), a permutation representation R of P in Figure 1(b), the transitive orientation Φ_R of P in Figure 1(c), and the corresponding simple directed graph F_R in Figure 1(d). In the figure, the pairs $\{u_i^1, u_i^2\}_{i=1}^3$ are grouped inside ellipses. In this example, R is not an acyclic permutation representation with respect to $\{u_i^1, u_i^2\}_{i=1}^3$, since F_R has a directed cycle of length two. However, note that, by exchanging the lines u_1^1 and u_2^1 in R, the resulting permutation representation R' is acyclic with respect to $\{u_i^1, u_i^2\}_{i=1}^3$, and thus P is acyclic with respect to $\{u_i^1, u_i^2\}_{i=1}^3$.

[*]To simplify the presentation, we use throughout the paper $\{u_i^1, u_i^2\}_{i=1}^n$ to denote the set of n unordered pairs $\{u_1^1, u_1^2\}, \{u_2^1, u_2^2\}, \ldots, \{u_n^1, u_n^2\}$.

Definition 2 *Let G be a trapezoid graph with n vertices and R be a trapezoid representation of G. Let P be the permutation graph with $2n$ vertices corresponding to the left and right lines of the trapezoids in R, R_P be the permutation representation of P induced by R, and $\{u_i^1, u_i^2\}$ be the vertices of P that correspond to the same vertex u_i of G, $i = 1, 2, \ldots, n$. Then,*

1. *R is an* acyclic trapezoid representation*, if R_P is an acyclic permutation representation with respect to $\{u_i^1, u_i^2\}_{i=1}^n$,*
2. *G is an* acyclic trapezoid graph*, if it has an acyclic representation R.*

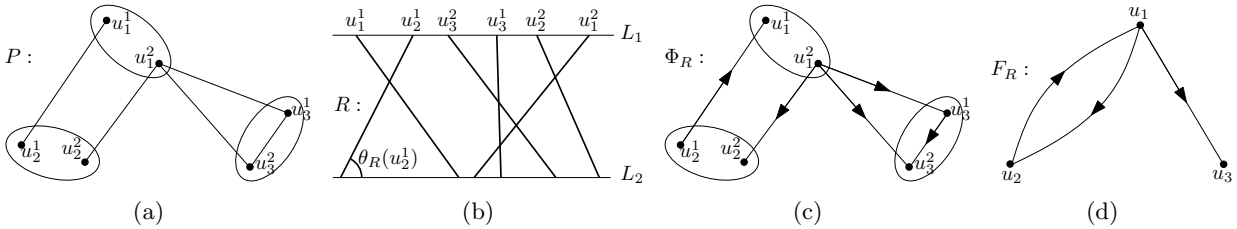

Figure 1: (a) A permutation graph P, (b) a permutation representation R of P, (c) the transitive orientation Φ_R of P, and (d) the corresponding simple directed graph F_R.

The following lemma follows easily from Definitions 1 and 2. The proof can be found in Appendix A.

Lemma 1 *Any parallelogram graph is an acyclic trapezoid graph.*

2.2 Structural properties of trapezoid graphs

In the following, we state some definitions concerning an arbitrary simple undirected graph $G = (V, E)$, which are useful for our analysis. Although these definitions apply to any graph, we will use them only for trapezoid graphs. Similar definitions, for the restricted case where the graph G is connected, were studied in [6]. For $u \in V$ and $U \subseteq V$, $N(u) = \{v \in V \mid uv \in E\}$ is the set of adjacent vertices of u in G, $N[u] = N(u) \cup \{u\}$, and $N(U) = \bigcup_{u \in U} N(u) \setminus U$. If $N(U) \subseteq N(W)$ for two vertex subsets U and W, then U is said to be *neighborhood dominated* by W. Clearly, the relationship of neighborhood domination is transitive.

Let $C_1, C_2, \ldots, C_\omega$, $\omega \geq 1$, be the connected components of $G \setminus N[u]$ and $V_i = V(C_i)$, $i = 1, 2, \ldots, \omega$. For simplicity of the presentation, we will identify in the sequel the component C_i and its vertex set V_i, $i = 1, 2, \ldots, \omega$. For $i = 1, 2, \ldots, \omega$, the *neighborhood domination closure* of V_i with respect to u is the set $D_u(V_i) = \{V_p \mid N(V_p) \subseteq N(V_i),\ p = 1, 2, \ldots, \omega\}$ of connected components of $G \setminus N[u]$. A component V_i is called a *master component* of u if $|D_u(V_i)| \geq |D_u(V_j)|$ for all $j = 1, 2, \ldots, \omega$. The *closure complement* of the neighborhood domination closure $D_u(V_i)$ is the set $D_u^*(V_i) = \{V_1, V_2, \ldots, V_\omega\} \setminus D_u(V_i)$. Finally, for a subset $S \subseteq \{V_1, V_2, \ldots, V_\omega\}$, a component $V_j \in S$ is called *maximal* if there is no component $V_k \in S$ such that $N(V_j) \subsetneqq N(V_k)$.

For example, consider the trapezoid graph G with vertex set $\{u, u_1, u_2, u_3, v_1, v_2, v_3, v_4\}$, which is given by the trapezoid representation R of Figure 2. The connected components of $G \setminus N[u] = \{v_1, v_2, v_3, v_4\}$ are $V_1 = \{v_1\}$, $V_2 = \{v_2\}$, $V_3 = \{v_3\}$, and $V_4 = \{v_4\}$. Then, $N(V_1) = \{u_1\}$, $N(V_2) = \{u_1, u_3\}$, $N(V_3) = \{u_2, u_3\}$, and $N(V_4) = \{u_3\}$. Hence, $D_u(V_1) = \{V_1\}$, $D_u(V_2) = \{V_1, V_2, V_4\}$, $D_u(V_3) = \{V_3, V_4\}$, and $D_u(V_4) = \{V_4\}$; thus, V_2 is the only master component of u. Furthermore, $D_u^*(V_1) = \{V_2, V_3, V_4\}$, $D_u^*(V_2) = \{V_3\}$, $D_u^*(V_3) = \{V_1, V_2\}$, and $D_u^*(V_4) = \{V_1, V_2, V_3\}$.

Lemma 2 *Let G be a simple graph, u be a vertex of G, and let $V_1, V_2, \ldots, V_\omega$, $\omega \geq 1$, be the connected components of $G \setminus N[u]$. If V_i is a master component of u, such that $D_u^*(V_i) \neq \emptyset$, then $D_u^*(V_j) \neq \emptyset$ for every component V_j of $G \setminus N[u]$.*

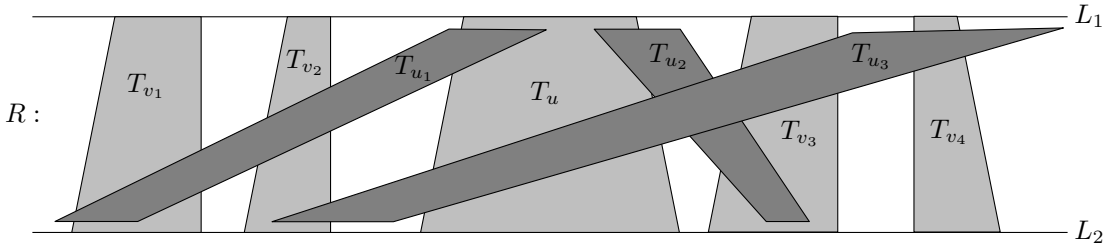

Figure 2: A trapezoid representation R of a trapezoid graph G.

Proof. Since V_i is a master component, and since $D_u^*(V_i) \neq \emptyset$, it follows that $|D_u(V_j)| \leq |D_u(V_i)| < \omega$ for every connected component $V_j \in \{V_1, V_2, \ldots, V_\omega\}$. Therefore, $|D_u(V_j)| < \omega$, and thus, $D_u^*(V_j) \neq \emptyset$ as well. ■

In the following we investigate several properties of trapezoid graphs, in order to derive the vertex-splitting algorithm Split-U in Section 2.3.

Remark 1 *Similar properties of trapezoid graphs have been studied in [6], leading to another vertex-splitting algorithm, called Split-All. However, the algorithm proposed in [6] is incorrect, since it is based on an incorrect property*[†]*, as was also verified by [7]. In the sequel of this section, we present new definitions and properties. In the cases where a similarity arises with those of [6], we refer to it specifically.*

The following lemma, which has been stated in Observation 3.1(4) in [6] (without a proof), will be used in our analysis below. For the sake of completeness, we present the proof in Appendix B.

Lemma 3 *Let R be a trapezoid representation of a trapezoid graph G, and V_i be a master component of a vertex u of G, such that $R(V_i) \ll_R T_u$. Then, $T_u \ll_R R(V_j)$ for every component $V_j \in D_u^*(V_i)$.*

Definition 3 *Let G be a trapezoid graph, u be a vertex of G, and V_i be an arbitrarily chosen master component of u. Then, $\delta_u = V_i$ and*

1. *if $D_u^*(V_i) = \emptyset$, then $\delta_u^* = \emptyset$.*
2. *if $D_u^*(V_i) \neq \emptyset$, then $\delta_u^* = V_j$, for an arbitrarily chosen maximal component $V_j \in D_u^*(V_i)$.*

Actually, as we will show in Lemma 4, the arbitrary choice of the components V_i and V_j in Definition 3 does not affect essentially the structural properties of G that we will investigate in the sequel. From now on, whenever we speak about δ_u and δ_u^*, we assume that these arbitrary choices of V_i and V_j have been already made.

Definition 4 *Let G be a trapezoid graph and u be a vertex of G. The vertices of $N(u)$ are partitioned into four possibly empty sets:*

1. *$N_0(u)$: vertices not adjacent to either δ_u or δ_u^*.*
2. *$N_1(u)$: vertices adjacent to δ_u but not to δ_u^*.*
3. *$N_2(u)$: vertices adjacent to δ_u^* but not to δ_u.*
4. *$N_{12}(u)$: vertices adjacent to both δ_u and δ_u^*.*

In the following definition we partition the neighbors of a vertex of a trapezoid graph G into four possibly empty sets. Note that these sets depend on a given trapezoid representation R of G, in contrast to the four sets of Definition 4 that depend only on the graph G itself.

[†]In Observation 3.1(5) of [6], it is claimed that for an arbitrary trapezoid representation R of a connected trapezoid graph G, where V_i is a master component of u such that $D_u^*(V_i) \neq \emptyset$ and $R(V_i) \ll_R T_u$, it holds $R(D_u(V_i)) \ll_R T_u \ll_R R(D_u^*(V_i))$. However, the first part of the latter inequality is not true. For instance, in the trapezoid graph G of Figure 2, $V_2 = \{v_2\}$ is a master component of u, where $D_u^*(V_2) = \{V_3\} = \{\{v_3\}\} \neq \emptyset$ and $R(V_2) \ll_R T_u$. However, $V_4 = \{v_4\} \in D_u(V_2)$ and $T_u \ll_R T_{v_4}$, and thus, $R(D_u(V_2)) \not\ll_R T_u$.

Definition 5 *Let G be a trapezoid graph, R be a representation of G, and u be a vertex of G. Denote by $D_1(u,R)$ and $D_2(u,R)$ the sets of trapezoids of R that lie completely to the left and to the right of T_u in R, respectively. Then, the vertices of $N(u)$ are partitioned into four possibly empty sets:*

1. *$N_0(u,R)$: vertices not adjacent to either $D_1(u,R)$ or $D_2(u,R)$.*
2. *$N_1(u,R)$: vertices adjacent to $D_1(u,R)$ but not to $D_2(u,R)$.*
3. *$N_2(u,R)$: vertices adjacent to $D_2(u,R)$ but not to $D_1(u,R)$.*
4. *$N_{12}(u,R)$: vertices adjacent to both $D_1(u,R)$ and $D_2(u,R)$.*

Suppose now that $\delta_u^* \neq \emptyset$, and let V_i be the master component of u that corresponds to δ_u, cf. Definition 3. Then, given any trapezoid representation R of G, we may assume w.l.o.g. that $R(V_i) \ll_R T_u$, by possibly performing a vertical axis flipping of R. The following lemma, which is proved in Appendix C, connects Definitions 4 and 5. The intuition behind this lemma is that the sets defined in Definition 4 include those neighbors of u, whose trapezoids intersect some trapezoids that lie to the left and/or to the right of T_u in R.

Lemma 4 *Let G be a trapezoid graph, R be a representation of G, and u be a vertex of G with $\delta_u^* \neq \emptyset$. Let V_i be the master component of u that corresponds to δ_u. If $R(V_i) \ll_R T_u$, then $N_X(u) = N_X(u,R)$ for every $X \in \{0,1,2,12\}$.*

2.3 A splitting algorithm

We define now the splitting of a vertex u of a trapezoid graph G, where $\delta_u^* \neq \emptyset$. Note that this splitting operation does not depend on any trapezoid representation of G. Intuitively, if the graph G was given along with a specific trapezoid representation R, this would have meant that we replace the trapezoid T_u in R by its two lines $l(T_u)$ and $r(T_u)$.

Definition 6 *Let G be a trapezoid graph and u be a vertex of G, where $\delta_u^* \neq \emptyset$. The graph $G^{\#}(u)$ obtained by the* vertex splitting *of u is defined as follows:*

1. *$V(G^{\#}(u)) = V(G) \setminus \{u\} \cup \{u_1, u_2\}$, where u_1 and u_2 are the two new vertices.*
2. *$E(G^{\#}(u)) = E[V(G) \setminus \{u\}] \cup \{u_1 x \mid x \in N_1(u)\} \cup \{u_2 x \mid x \in N_2(u)\} \cup \{u_1 x, u_2 x \mid x \in N_{12}(u)\}$.*

The vertices u_1 and u_2 are the derivatives *of vertex u.*

We state now the notion of a standard trapezoid representation with respect to a particular vertex.

Definition 7 *Let G be a trapezoid graph and u be a vertex of G, where $\delta_u^* \neq \emptyset$. A trapezoid representation R of G is* standard with respect to u, *if the following properties are satisfied:*

1. *$l(T_u) \ll_R R(N_0(u) \cup N_2(u))$.*
2. *$R(N_0(u) \cup N_1(u)) \ll_R r(T_u)$.*

Now, given a trapezoid graph G and a vertex subset $U = \{u_1, u_2, \ldots, u_k\}$, such that $\delta_{u_i}^* \neq \emptyset$ for every $i = 1, 2, \ldots, k$, Algorithm Split-U returns a graph $G^{\#}(U)$ by splitting every vertex of U exactly once. At every step, Algorithm Split-U splits a vertex of U, and finally, it removes all vertices of the set $V(G) \setminus U$, which have not been split.

Remark 2 *As mentioned in Remark 1, a similar algorithm, called Split-All, was presented in [6]. We would like to emphasize here the following four differences between the two algorithms. First, that Split-All gets as input a sibling-free graph G (two vertices u, v of a graph G are called* siblings, *if $N[u] = N[v]$; G is called* sibling-free *if G has no pair of sibling vertices), while our Algorithm Split-U gets as an input any graph (though, we will use it only for trapezoid graphs), which may*

Algorithm 1 Split-U

Input: A trapezoid graph G and a vertex subset $U = \{u_1, u_2, \ldots, u_k\}$, such that $\delta^*_{u_i} \neq \emptyset$ for all $i = 1, 2, \ldots, k$

Output: The permutation graph $G^{\#}(U)$

$\overline{U} \leftarrow V(G) \setminus U$; $H_0 \leftarrow G$

for $i = 1$ to k **do**

$H_i \leftarrow H^{\#}_{i-1}(u_i)$ {H_i is obtained by the vertex splitting of u_i in H_{i-1}}

$G^{\#}(U) \leftarrow H_k[V(H_k) \setminus \overline{U}]$ {remove from H_k all unsplitted vertices}

return $G^{\#}(U)$

contain pairs of sibling vertices. Second, Split-All splits all the vertices of the input graph, while Split-U splits only a subset of them, which satisfy a special property. Third, the order of vertices that are split by Split-All depends on a certain property (inclusion-minimal neighbor set), while Split-U splits the vertices in an arbitrary order. Last, the main difference between these two algorithms is that they perform a different vertex splitting operation at every step, since Definitions 3 and 4 do not comply with the corresponding Definitions 4.1 and 4.2 of [6].

The proof of the following theorem can be found in Appendix D.

Theorem 1 *Let G be a trapezoid graph and $U = \{u_1, u_2, \ldots, u_k\}$ be a vertex subset of G, such that $\delta^*_{u_i} \neq \emptyset$ for every $i = 1, 2, \ldots, k$. Then, the graph $G^{\#}(U)$ obtained by Algorithm Split-U, is a permutation graph with $2k$ vertices. Furthermore, if G is acyclic, then $G^{\#}(U)$ is acyclic with respect to $\{u_i^1, u_i^2\}_{i=1}^k$, where u_i^1 and u_i^2 are the derivatives of u_i, $i = 1, 2, \ldots, k$.*

3 The recognition of bounded tolerance graphs

In this section we provide a reduction from the *monotone-Not-All-Equal-3-SAT (monotone-NAE-3-SAT)* problem to the problem of recognizing whether a given graph is a bounded tolerance graph. A boolean formula ϕ is called *monotone* if no variable in ϕ is negated. Given a monotone boolean formula ϕ in conjunctive normal form with three literals in each clause (3-CNF), ϕ is *NAE-satisfiable* if there is a truth assignment of ϕ, such that every clause contains at least one true literal and at least one false one. The problem of deciding whether a given monotone 3-CNF formula ϕ is NAE-satisfiable is known to be NP-complete (see [29] for the NP-completeness of NAE-3-SAT[‡]). We can assume w.l.o.g. that each clause has three distinct literals, i.e. variables. Given a monotone 3-CNF formula ϕ, we construct in polynomial time a trapezoid graph H_ϕ, such that H_ϕ is a bounded tolerance graph if and only if ϕ is NAE-satisfiable. To this end, we construct first a permutation graph P_ϕ and a trapezoid graph G_ϕ.

3.1 The permutation graph P_ϕ

Consider a monotone 3-CNF formula $\phi = \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_k$ with k clauses and n boolean variables $x_1, x_2, \ldots, x_n$, such that $\alpha_i = (x_{r_{i,1}} \vee x_{r_{i,2}} \vee x_{r_{i,3}})$ for $i = 1, 2, \ldots, k$, where $1 \leq r_{i,1} < r_{i,2} < r_{i,3} \leq n$. We construct the permutation graph P_ϕ, along with a permutation representation R_P of P_ϕ, as follows. Let L_1 and L_2 be two parallel lines and let $\theta(\ell)$ denote the angle of the line ℓ with L_2 in R_P. For every clause α_i, $i = 1, 2, \ldots, k$, we correspond to each of the literals, i.e. variables, $x_{r_{i,1}}$, $x_{r_{i,2}}$, and $x_{r_{i,3}}$ a pair of intersecting lines with endpoints on L_1 and L_2. Namely, we correspond to the variable $x_{r_{i,1}}$ the pair $\{a_i, c_i\}$, to $x_{r_{i,2}}$ the pair $\{e_i, b_i\}$ and to $x_{r_{i,3}}$ the pair $\{d_i, f_i\}$, respectively, such that $\theta(a_i) > \theta(c_i)$, $\theta(e_i) > \theta(b_i)$, $\theta(d_i) > \theta(f_i)$, and such that the lines a_i, c_i lie completely to the left of e_i, b_i in R_P, and e_i, b_i lie completely to the left of d_i, f_i in R_P, as it is illustrated in Figure 3.

[‡]To reduce NAE-3-SAT to monotone-NAE-3-SAT, replace each variable x with two variables x_0 and x_1 (depending on whether x appears negated or not), add variables x_2, x_3, x_4, and add the clauses $(x_0 \vee x_1 \vee x_2)$, $(x_0 \vee x_1 \vee x_3)$, $(x_0 \vee x_1 \vee x_4)$, and $(x_2 \vee x_3 \vee x_4)$.

Denote the lines that correspond to the variable $x_{r_{i,j}}$, $j = 1, 2, 3$, by $\ell^1_{i,j}$ and $\ell^2_{i,j}$, respectively, such that $\theta(\ell^1_{i,j}) > \theta(\ell^2_{i,j})$. That is, $(\ell^1_{i,1}, \ell^2_{i,1}) = (a_i, c_i)$, $(\ell^1_{i,2}, \ell^2_{i,2}) = (e_i, b_i)$, and $(\ell^1_{i,3}, \ell^2_{i,3}) = (d_i, f_i)$. Note that no line of a pair $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ intersects with a line of another pair $\{\ell^1_{i',j'}, \ell^2_{i',j'}\}$.

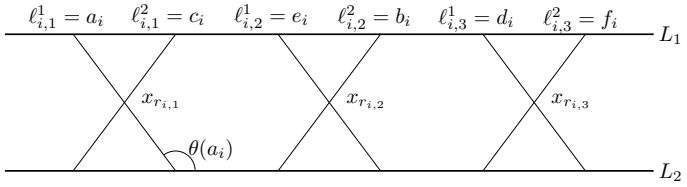

Figure 3: The six lines of the permutation graph P_ϕ, which correspond to the clause $\alpha_i = (x_{r_{i,1}} \vee x_{r_{i,2}} \vee x_{r_{i,3}})$ of the boolean formula ϕ.

Denote by S_p, $p = 1, 2, \ldots, n$, the set of pairs $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ that correspond to the variable x_p, i.e. $r_{i,j} = p$. We order the pairs $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ such that any pair of S_{p_1} lies completely to the left of any pair of S_{p_2}, whenever $p_1 < p_2$, while the pairs that belong to the same set S_p are ordered arbitrarily. For two consecutive pairs $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ and $\{\ell^1_{i',j'}, \ell^2_{i',j'}\}$ in S_p, where $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ lies to the left of $\{\ell^1_{i',j'}, \ell^2_{i',j'}\}$, we add a pair $\{u^{i',j'}_{i,j}, v^{i',j'}_{i,j}\}$ of parallel lines that intersect both $\ell^1_{i,j}$ and $\ell^1_{i',j'}$, but no other line. Note that $\theta(\ell^1_{i,j}) > \theta(u^{i',j'}_{i,j})$ and $\theta(\ell^1_{i',j'}) > \theta(u^{i',j'}_{i,j})$, while $\theta(u^{i',j'}_{i,j}) = \theta(v^{i',j'}_{i,j})$. This completes the construction. Denote the resulting permutation graph by P_ϕ, and the corresponding permutation representation of P_ϕ by R_P. Observe that P_ϕ has n connected components, which are called *blocks*, one for each variable $x_1, x_2, \ldots, x_n$.

An example of the construction of P_ϕ and R_P from ϕ with $k = 3$ clauses and $n = 4$ variables is illustrated in Figure 4. In this figure, the lines $u^{i',j'}_{i,j}$ and $v^{i',j'}_{i,j}$ are drawn in bold.

The formula ϕ has $3k$ literals, and thus the permutation graph P_ϕ has $6k$ lines $\ell^1_{i,j}, \ell^2_{i,j}$ in R_P, one pair for each literal. Furthermore, two lines $u^{i',j'}_{i,j}, v^{i',j'}_{i,j}$ correspond to each pair of consecutive pairs $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ and $\{\ell^1_{i',j'}, \ell^2_{i',j'}\}$ in R_P, except for the case where these pairs of lines belong to different variables, i.e. when $r_{i,j} \neq r_{i',j'}$. Therefore, since ϕ has n variables, there are $2(3k - n) = 6k - 2n$ lines $u^{i',j'}_{i,j}, v^{i',j'}_{i,j}$ in R_P. Thus, R_P has in total $12k - 2n$ lines, i.e. P_ϕ has $12k - 2n$ vertices. In the example of Figure 4, $k = 3$, $n = 4$, and thus, P_ϕ has 28 vertices.

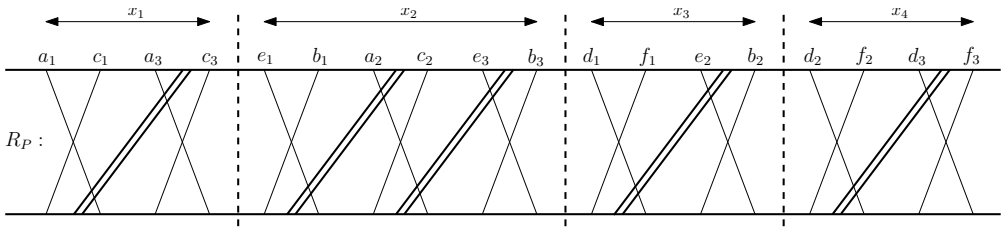

Figure 4: The permutation representation R_P of the permutation graph P_ϕ for $\phi = \alpha_1 \wedge \alpha_2 \wedge \alpha_3 = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4)$.

Let $m = 6k - n$, where $2m$ is the number of vertices in P_ϕ. We group the lines of R_P, i.e. the vertices of P_ϕ, into pairs $\{u^1_i, u^2_i\}^m_{i=1}$, as follows. For every clause α_i, $i = 1, 2, \ldots, k$, we group the lines $a_i, b_i, c_i, d_i, e_i, f_i$ into the three pairs $\{a_i, b_i\}$, $\{c_i, d_i\}$, and $\{e_i, f_i\}$. The remaining lines are grouped naturally according to the construction; namely, every two lines $\{u^{i',j'}_{i,j}, v^{i',j'}_{i,j}\}$ constitute a pair. The proof of the following lemma can be found in Appendix E.

Lemma 5 *If the permutation graph P_ϕ is acyclic with respect to $\{u^1_i, u^2_i\}^m_{i=1}$ then the formula ϕ is NAE-satisfiable.*

For the formula ϕ of Figure 4, an example of an acyclic permutation representation R_0 of P_ϕ with respect to $\{u^1_i, u^2_i\}^m_{i=1}$, along with the corresponding transitive orientation Φ_{R_0} of P_ϕ,

is illustrated in Figure 5. This transitive orientation corresponds to the NAE-satisfying truth assignment $(x_1, x_2, x_3, x_4) = (1, 1, 0, 0)$ of ϕ. Similarly to Figure 4, the lines $u_{i,j}^{i',j'}$ and $v_{i,j}^{i',j'}$ are drawn in bold in Figure 5(a). Furthermore, for better visibility, the vertices that correspond to these lines are grouped in shadowed ellipses in Figure 5(b), while the arcs incident to them are drawn dashed.

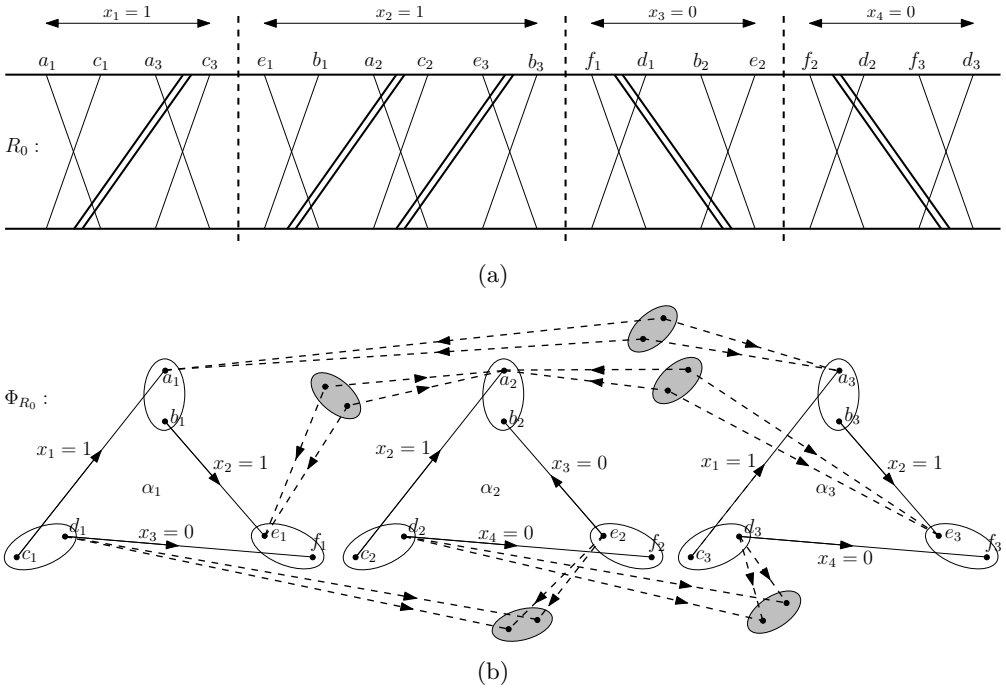

Figure 5: The NAE-satisfying truth assignment $(x_1, x_2, x_3, x_4) = (1, 1, 0, 0)$ of the formula ϕ of Figure 4: (a) an acyclic permutation representation R_0 of P_ϕ and (b) the corresponding transitive orientation Φ_{R_0} of P_ϕ.

3.2 The trapezoid graphs G_ϕ and H_ϕ

Let $\{u_i^1, u_i^2\}_{i=1}^m$ be the pairs of vertices in the constructed permutation graph P_ϕ and R_P be its permutation representation. We construct now from P_ϕ the trapezoid graph G_ϕ with m vertices $\{u_1, u_2, \ldots, u_m\}$, as follows. We replace in the permutation representation R_P for every $i = 1, 2, \ldots, m$ the lines u_i^1 and u_i^2 by the trapezoid T_{u_i}, which has u_i^1 and u_i^2 as its left and right lines, respectively. Let R_G be the resulting trapezoid representation of G_ϕ.

Finally, we construct from G_ϕ the trapezoid graph H_ϕ with $7m$ vertices, by adding to every trapezoid T_{u_i}, $i = 1, 2, \ldots, m$, six parallelograms $T_{u_{i,1}}, T_{u_{i,2}}, \ldots, T_{u_{i,6}}$ in the trapezoid representation R_G, as follows. Let ε be the smallest distance in R_G between two different endpoints on L_1, or on L_2. The right (resp. left) line of $T_{u_{1,1}}$ lies to the right (resp. left) of u_1^1, and it is parallel to it at distance $\frac{\varepsilon}{2}$. The right (resp. left) line of $T_{u_{1,2}}$ lies to the left of u_1^1, and it is parallel to it at distance $\frac{\varepsilon}{4}$ (resp. $\frac{3\varepsilon}{4}$). Moreover, the right (resp. left) line of $T_{u_{1,3}}$ lies to the left of u_1^1, and it is parallel to it at distance $\frac{3\varepsilon}{8}$ (resp. $\frac{7\varepsilon}{8}$). Similarly, the left (resp. right) line of $T_{u_{1,4}}$ lies to the left (resp. right) of u_1^2, and it is parallel to it at distance $\frac{\varepsilon}{2}$. The left (resp. right) line of $T_{u_{1,5}}$ lies to the right of u_1^2, and it is parallel to it at distance $\frac{\varepsilon}{4}$ (resp. $\frac{3\varepsilon}{4}$). Finally, the right (resp. left) line of $T_{u_{1,6}}$ lies to the right of u_1^2, and it is parallel to it at distance $\frac{3\varepsilon}{8}$ (resp. $\frac{7\varepsilon}{8}$), as illustrated in Figure 6.

After adding the parallelograms $T_{u_{1,1}}, T_{u_{1,2}}, \ldots, T_{u_{1,6}}$ to a trapezoid T_{u_1}, we update the smallest distance ε between two different endpoints on L_1, or on L_2 in the resulting representation, and we continue the construction iteratively for all $i = 2, \ldots, m$. Denote by H_ϕ the resulting trapezoid graph with $7m$ vertices, and by R_H the corresponding trapezoid representation. Note that in R_H, between the endpoints of the parallelograms $T_{u_{i,1}}$, $T_{u_{i,2}}$, and $T_{u_{i,3}}$ (resp. $T_{u_{i,4}}$, $T_{u_{i,5}}$, and $T_{u_{i,6}}$) on L_1

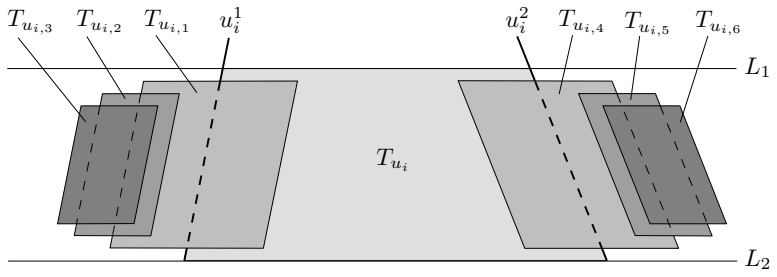

Figure 6: The addition of the six parallelograms $T_{u_i,1}, T_{u_i,2}, \ldots, T_{u_i,6}$ to the trapezoid T_{u_i}, $i = 1, 2, \ldots, m$, in the construction of the trapezoid graph H_ϕ from G_ϕ.

and L_2, there are no other endpoints of H_ϕ, except those of u_i^1 (resp. u_i^2), for every $i = 1, 2, \ldots, m$. Furthermore, note that R_H is standard with respect to u_i, for every $i = 1, 2, \ldots, m$. The following auxiliary lemma is crucial in the proof of Theorem 2. The proof of Lemma 6 and Theorem 2 can be found in Appendices F and G, respectively.

Lemma 6 *In the trapezoid graph H_ϕ, $\delta^*_{u_i} \neq \emptyset$ for every $i = 1, 2, \ldots, m$.*

Theorem 2 *The formula ϕ is NAE-satisfiable if and only if the trapezoid graph H_ϕ is a bounded tolerance graph.*

Therefore, since monotone-NAE-3-SAT is NP-complete, the problem of recognizing bounded tolerance graphs is NP-hard. Moreover, since the recognition of bounded tolerance graphs lies in NP [17], we can summarize our results as follows.

Theorem 3 *Given a graph G, it is NP-complete to decide whether it is a bounded tolerance graph.*

4 The recognition of tolerance graphs

In this section we show that the reduction from the monotone-NAE-3-SAT problem to the problem of recognizing bounded tolerance graphs presented in Section 3, can be extended to the problem of recognizing general tolerance graphs. In particular, we prove that:

Lemma 7 *If H_ϕ is a tolerance graph then it is a bounded tolerance graph.*

Using this lemma, we prove the main result of this section. Due to space limitations, the details can be found in Appendix H.

Theorem 4 *Given a graph G, it is NP-complete to decide whether it is a tolerance graph. The problem remains NP-complete even if the given graph G is known to be a trapezoid graph.*

5 Concluding remarks

In this article we proved that both tolerance and bounded tolerance graph recognition problems are NP-complete, by providing a reduction from the monotone-NAE-3-SAT problem, thus answering a longstanding open question. Furthermore, our reduction implies that, given a trapezoid graph, it is NP-complete to decide whether this graph is a tolerance or a bounded tolerance (i.e. parallelogram) graph. A *unit* interval representation is an interval representation in which all intervals have the same length. A *proper* interval representation is one in which no interval is properly contained in another. These terms can apply to both interval graphs and tolerance graphs. It is known that the subclasses of unit and proper interval graphs are equal [27], but the corresponding tolerance subclasses are different [2]. The recognition of unit and of proper tolerance graphs, as well as of any other subclass of tolerance graphs, except bounded tolerance and bipartite tolerance graphs [5], remain interesting open problems [14].

References

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[2] K. P. Bogart, P. C. Fishburnb, G. Isaak, and L. Langley. Proper and unit tolerance graphs. *Discrete Applied Mathematics*, 60(1-3):99–117, 1995.

[3] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph classes: a survey*. Society for Industrial and Applied Mathematics, 1999.

[4] A. H. Busch. A characterization of triangle-free tolerance graphs. *Discrete Applied Mathematics*, 154(3):471–477, 2006.

[5] A. H. Busch and G. Isaak. Recognizing bipartite tolerance graphs in linear time. In *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 12–20, 2007.

[6] F. Cheah and D. Corneil. On the structure of trapezoid graphs. *Discrete Applied Mathematics*, 66(2):109–133, 1996.

[7] F. Cheah and D. Corneil, 2009. Personal communication.

[8] S. Felsner. Tolerance graphs and orders. *Journal of Graph Theory*, 28:129–140, 1998.

[9] P. Fishburn and W. Trotter. Split semiorders. *Discrete Mathematics*, 195:111–126, 1999.

[10] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol. 57)*. North-Holland Publishing Co., 2004.

[11] M. Golumbic and C. Monma. A generalization of interval graphs with tolerances. In *Proceedings of the 13th Southeastern Conference on Combinatorics, Graph Theory and Computing, Congressus Numerantium 35*, pages 321–331, 1982.

[12] M. Golumbic, C. Monma, and W. Trotter. Tolerance graphs. *Discrete Applied Mathematics*, 9(2):157–170, 1984.

[13] M. Golumbic and A. Siani. Coloring algorithms for tolerance graphs: Reasoning and scheduling with interval constraints. In *Proceedings of the Joint International Conferences on Artificial Intelligence, Automated Reasoning, and Symbolic Computation (AISC/Calculemus)*, pages 196–207, 2002.

[14] M. Golumbic and A. Trenk. *Tolerance Graphs*. Cambridge Studies in Advanced Mathematics, 2004.

[15] M. C. Golumbic and R. E. Jamison. Edge and vertex intersection of paths in a tree. *Discrete Mathematics*, 55(2):151–159, 1985.

[16] M. Grötshcel, L. Lovász, and A. Schrijver. The Ellipsoid Method and its Consequences in Combinatorial Optimization. *Combinatorica*, 1:169–197, 1981.

[17] R. B. Hayward and R. Shamir. A note on tolerance graph recognition. *Discrete Applied Mathematics*, 143(1-3):307–311, 2004.

[18] G. Isaak, K. Nyman, and A. Trenk. A hierarchy of classes of bounded bitolerance orders. *Ars Combinatoria*, 69, 2003.

[19] M. Kaufmann, J. Kratochvíl, K. A. Lehmann, and A. R. Subramanian. Max-tolerance graphs as intersection graphs: cliques, cycles, and recognition. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 832–841, 2006.

[20] J. M. Keil and P. Belleville. Dominating the complements of bounded tolerance graphs and the complements of trapezoid graphs. *Discrete Applied Mathematics*, 140(1-3):73–89, 2004.

[21] L. Langley. *Interval tolerance orders and dimension*. PhD thesis, Dartmouth College, 1993.

[22] T.-H. Ma and J. P. Spinrad. On the 2-chain subgraph cover and related problems. *Journal of Algorithms*, 17(2):251–268, 1994.

[23] G. B. Mertzios and D. G. Corneil. Vertex splitting and the recognition of trapezoid graphs, 2009. Manuscript in preparation.

[24] G. B. Mertzios, I. Sau, and S. Zaks. A new intersection model and improved algorithms for tolerance graphs. In *Proceedings of the 35rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 2009. To appear.

[25] M. Middendorf and F. Pfeiffer. On the complexity of recognizing perfectly orderable graphs. *Discrete Mathematics*, 80(3):327–333, 1990.

[26] G. Narasimhan and R. Manber. Stability and chromatic number of tolerance graphs. *Discrete Applied Mathematics*, 36:47–56, 1992.

[27] F. Roberts. *Indifference graphs*. Proof Techniques in Graph Theory, Academic Press, New York, 139-146, 1969.

[28] S. P. Ryan. Trapezoid order classification. *Order*, 15:341–354, 1998.

[29] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th annual ACM symposium on Theory of computing (STOC)*, pages 216–226, 1978.

[30] J. P. Spinrad. *Efficient graph representations*, volume 19 of *Fields Institute Monographs*. American Mathematical Society, 2003.

Appendix A: Proof of Lemma 1

Let G be a parallelogram graph with n vertices $\{u_1, u_2, \ldots, u_n\}$ and R be a parallelogram representation of G. That is, R is a trapezoid representation of G, such that the left and right lines $l(T_{u_i})$ and $r(T_{u_i})$ of the trapezoid T_{u_i}, $i = 1, 2, \ldots, n$, are parallel in R, i.e. $\theta_R(l(T_{u_i})) = \theta_R(r(T_{u_i}))$. Let P be the permutation graph with $2n$ vertices $\{u_1^1, u_1^2, u_2^1, u_2^2, \ldots, u_n^1, u_n^2\}$ corresponding to the left and right lines of the trapezoids of G in R, i.e. the vertices u_i^1 and u_i^2 correspond to $l(T_{u_i})$ and $r(T_{u_i})$, $i = 1, 2, \ldots, n$, respectively. Let R_P be the permutation representation of P induced by R, and Φ_{R_P} be the corresponding transitive orientation of the permutation graph P. Recall that, for two intersecting lines a, b in R_P, it holds $ab \in \Phi_{R_P}$ whenever $\theta_R(a) < \theta_R(b)$. It follows that for any $i = 1, 2, \ldots, n$, the pair $\{u_i^1, u_i^2\}$ of vertices in P has incoming edges from (resp. outgoing edges to) vertices of other pairs $\{u_j^1, u_j^2\}$ in Φ_{R_P}, which have smaller (resp. greater) angle with the line L_2 in R_P. Thus, the simple directed graph F_{R_P} defined in Definition 1 has no directed cycles, and therefore R_P is an acyclic permutation representation with respect to $\{u_i^1, u_i^2\}_{i=1}^n$, i.e. R is an acyclic trapezoid representation of G by Definition 2. ■

Appendix B: Proof of Lemma 3

Suppose otherwise that $R(V_j) \ll_R T_u$, for some $V_j \in D_u^*(V_i)$. Consider first the case where $R(V_j) \ll_R R(V_i) \ll_R T_u$. Then, since V_i lies between V_j and T_u in R, all trapezoids that intersect T_u and V_j, must intersect also V_i. Thus, $N(V_j) \subseteq N(V_i)$, i.e. $V_j \in D_u(V_i)$, which is a contradiction, since $V_j \in D_u^*(V_i)$. Consider now the case where $R(V_i) \ll_R R(V_j) \ll_R T_u$. Then, we obtain similarly that $N(V_i) \subseteq N(V_j)$, and thus, $D_u(V_i) \subseteq D_u(V_j)$. Since $V_j \in D_u(V_j) \setminus D_u(V_i)$, it follows that $|D_u(V_i)| < |D_u(V_j)|$. This is a contradiction to the assumption that V_i is a master component of u. Thus, $T_u \ll_R R(V_j)$ for any $V_j \in D_u^*(V_i)$. ■

Appendix C: Proof of Lemma 4

In order to prove Lemma 4, we first partition the neighbors of a vertex in a trapezoid graph G into four possibly empty sets. Note that these sets depend only on the graph G itself, in contrast to the four sets of Definition 5, which depend on a particular trapezoid representation of G.

Definition 8 *Let G be a trapezoid graph, and u be a vertex of G. Let V_i be a master component of u, such that $D_u^*(V_i) \neq \emptyset$, and V_j be a maximal component of $D_u^*(V_i)$. Then, the vertices of $N(u)$ are partitioned into four possibly empty sets:*

1. *$N_0(u, V_i, V_j)$: vertices not adjacent to either V_i or V_j.*
2. *$N_1(u, V_i, V_j)$: vertices adjacent to V_i but not to V_j.*
3. *$N_2(u, V_i, V_j)$: vertices adjacent to V_j but not to V_i.*
4. *$N_{12}(u, V_i, V_j)$: vertices adjacent to both V_i and V_j.*

The following lemma connects Definitions 5 and 8. The intuition behind this lemma is that the sets defined in Definition 8 include those neighbors of u, whose trapezoids intersect some trapezoids that lie to the left and/or to the right of T_u in a trapezoid representation of G.

Lemma 8 *Let G be a trapezoid graph, R be a representation of G, and u be a vertex of G. Let V_i be a master component of u, such that $D_u^*(V_i) \neq \emptyset$, and let V_j be a maximal component of $D_u^*(V_i)$. If $R(V_i) \ll_R T_u$, then $N_X(u, V_i, V_j) = N_X(u, R)$ for every $X \in \{0, 1, 2, 12\}$.*

Proof. Since $D_u^*(V_i) \neq \emptyset$ and $R(V_i) \ll_R T_u$, it follows by Lemma 3 that $T_u \ll_R R(V_j)$, i.e. $V_j \in D_2(u, R)$. Suppose that a component $V_\ell \neq V_j$ is the leftmost one of $D_2(u, R)$ in R,

i.e. $T_u \ll_R R(V_\ell) \ll_R R(V_j)$. Since V_ℓ lies between T_u and V_j in R, all trapezoids that intersect T_u and V_j, must also intersect V_ℓ, and thus, $N(V_j) \subseteq N(V_\ell)$. It follows that $V_\ell \in D_u^*(V_i)$, i.e. $V_\ell \notin D_u(V_i)$, since otherwise $V_j \in D_u(V_i)$, which is a contradiction. Furthermore, since V_j is a maximal component of $D_u^*(V_i)$, and since $N(V_j) \subseteq N(V_\ell)$, it follows that $N(V_j) = N(V_\ell)$, i.e. $N_X(u, V_i, V_j) = N_X(u, V_i, V_\ell)$ for every $X \in \{0, 1, 2, 12\}$.

Suppose that a component $V_k \neq V_i$ is the rightmost one of $D_1(u, R)$ in R, i.e. $R(V_i) \ll_R R(V_k) \ll_R T_u$. Then, $V_k \in D_u(V_i)$, since otherwise $T_u \ll_R R(V_k)$ by Lemma 3, which is a contradiction. Thus, $N(V_k) \subseteq N(V_i)$. Furthermore, since V_k lies between V_j and T_u in R, all trapezoids that intersect T_u and V_j, must also intersect V_k, and thus, $N(V_i) \subseteq N(V_k)$. Therefore, $N(V_i) = N(V_k)$, i.e. $N_X(u, V_i, V_\ell) = N_X(u, V_k, V_\ell)$ for every $X \in \{0, 1, 2, 12\}$, and thus, $N_X(u, V_i, V_j) = N_X(u, V_k, V_\ell)$ for every $X \in \{0, 1, 2, 12\}$.

Consider now a vertex $v \in N(u)$, and recall that V_k (resp. V_ℓ) is the rightmost (resp. leftmost) component of $D_1(u, R)$ (resp. $D_2(u, R)$) in R. Thus, if T_v intersects at least one component of $D_1(u, R)$ (resp. $D_2(u, R)$), then T_v intersects also with V_k (resp. V_ℓ). On the other hand, if T_v does not intersect any component of $D_1(u, R)$ (resp. $D_2(u, R)$), then T_v clearly does not intersect V_k (resp. V_ℓ), since $V_k \subseteq D_1(u, R)$ (resp. $V_j \subseteq D_2(u, R)$). It follows that $N_X(u, V_k, V_\ell) = N_X(u, R)$, and thus, $N_X(u, V_i, V_j) = N_X(u, R)$ for every $X \in \{0, 1, 2, 12\}$. This proves the lemma. ■

We now return to the proof of Lemma 4.

Proof of Lemma 4. Suppose that $\delta_u^* \neq \emptyset$, and let V_i be the master component of u and V_j be the maximal component of $D_u^*(V_i)$, which correspond to δ_u and δ_u^*, cf. Definition 3. Then, given a trapezoid representation R of G, we may assume in Lemma 8 w.l.o.g. that $R(V_i) \ll_R T_u$, by possibly performing a vertical axis flipping of R. Thus, since $N_X(u) = N_X(u, V_i, V_j)$ for every $X \in \{0, 1, 2, 12\}$, Lemma 4 follows from Lemma 8. ■

Appendix D: Proof of Theorem 1

In order to prove Theorem 1, we present first two auxiliary lemmas.

Lemma 9 *Let G be a trapezoid graph and u be a vertex of G. Then, $N_2(u) \cup N_{12}(u) = \emptyset$ if and only if $\delta_u^* = \emptyset$.*

Proof. Suppose first that $\delta_u^* = \emptyset$. Then, clearly there exists no vertex $v \in N(u)$ adjacent to δ_u^*, and thus, $N_2(u) \cup N_{12}(u) = \emptyset$. Conversely, suppose that $N_2(u) \cup N_{12}(u) = \emptyset$, and assume that $\delta_u^* \neq \emptyset$. Let $\delta_u = V_i$ and $\delta_u^* = V_j$, where V_i is a master component of u and V_j is a maximal component of $D_u^*(V_i)$. If $N(V_j) = \emptyset$, then clearly $N(V_j) \subseteq N(V_i)$, and thus, $V_j \in D_u(V_i)$, which is a contradiction. Thus, $N(V_j) \neq \emptyset$, i.e. some vertices of $N(u)$ are adjacent to some vertices of V_j. Since $\delta_u^* = V_j$, it follows by Definition 4 that $N_2(u) \cup N_{12}(u) \neq \emptyset$, which is a contradiction. Thus, $\delta_u^* = \emptyset$. ■

Lemma 10 *Let G be a trapezoid graph and u be a vertex of G. If $\delta_u^* \neq \emptyset$, then $N_1(u) \cup N_{12}(u) \neq \emptyset$.*

Proof. Suppose that $\delta_u^* \neq \emptyset$. Let V_i be the master component that corresponds to δ_u, and V_j be the maximal component of $D_u^*(V_i)$ that corresponds to δ_u^*. Assume that $N_1(u) \cup N_{12}(u) = \emptyset$, i.e. no neighbor of u is adjacent to any vertex $v \in V_i$. It follows that $N(V_i) = \emptyset$. On the other hand, since $\delta_u^* \neq \emptyset$, we obtain by Lemma 9 that $N_2(u) \cup N_{12}(u) \neq \emptyset$. That is, some neighbors of u are adjacent to some vertices of V_j, i.e. $N(V_j) \neq \emptyset$. Therefore, $N(V_i) = \emptyset \subsetneqq N(V_j)$, and thus, $D_u(V_i) \subsetneqq D_u(V_j)$, i.e. $|D_u(V_i)| < |D_u(V_j)|$. This is a contradiction, since V_i is a master component of u. Thus, $N_1(u) \cup N_{12}(u) \neq \emptyset$. ■

We are now ready to provide the proof of Theorem 1.

Proof of Theorem 1. Let R be a trapezoid representation of G. In order to prove that the graph $G^\#(U)$ constructed by Algorithm Split-All is a permutation graph, we will construct from R a permutation representation $R^\#(U)$ of $G^\#(U)$. To this end, we will construct sequentially, for

every $i = 1, 2, \ldots, k$, a standard trapezoid representation of H_{i-1} with respect to u_i, in which all derivatives u_j^1, u_j^2, $1 \leq j \leq i-1$, are represented by trivial trapezoids, i.e. lines.

Let $u = u_1$. If R is not a standard representation with respect to u, we construct first from R a trapezoid representation R' of G that satisfies the first condition of Definition 7. Then, we construct from R' a trapezoid representation R'' of G that satisfies also the second condition of Definition 7, i.e. R'' is a standard trapezoid representation R' of G with respect to u.

Let V_i be the master component of u that corresponds to δ_u. By possibly performing a vertical axis flipping of R, we may assume w.l.o.g. that $R(V_i) \ll_R T_u$. Furthermore, the sets $N_0(u)$, $N_1(u)$, $N_2(u)$, and $N_{12}(u)$ coincide by Lemma 4 with the sets $N_0(u, R)$, $N_1(u, R)$, $N_2(u, R)$, and $N_{12}(u, R)$, respectively. Recall that, by Definition 5, $D_1(u, R)$ and $D_2(u, R)$ denote the sets of trapezoids of R that lie completely to the left and to the right of T_u in R, respectively.

Let p_x and q_x the endpoints on L_1 and L_2, respectively, of the left line $l(T_x)$ of an arbitrary trapezoid T_x in R. Suppose that $N_0(u) \cup N_2(u) \neq \emptyset$. Let p_v and q_w be the leftmost endpoints on L_1 and L_2, respectively, of the trapezoids of $N_0(u) \cup N_2(u)$, and suppose that $p_v < p_u$ and $q_w < q_u$. Note that, possibly, $v = w$. Then, all vertices x, for which T_x has an endpoint between p_v and p_u on L_1 (resp. between q_w and q_u on L_2) are adjacent to u. Indeed, suppose otherwise that $T_x \cap T_u = \emptyset$, for such a vertex x. Then, since $T_v \cap T_u \neq \emptyset$ (resp. $T_w \cap T_u \neq \emptyset$), it follows that $T_x \cap T_v \neq \emptyset$ (resp. $T_x \cap T_w \neq \emptyset$). However, since $T_x \cap T_u = \emptyset$ and T_x has an endpoint to the left of T_u in R, it follows that $T_x \ll_R T_u$, i.e. $x \in D_1(u, R)$, and thus, by Definition 8, $v \in N_1(u) \cup N_{12}(u)$ (resp. $w \in N_1(u) \cup N_{12}(u)$), which is a contradiction.

Consider now a vertex $z \in N_1(u) \cup N_{12}(u)$ with $l(T_z) \ll_R l(T_u)$, where $p_v < p_z < p_u$. Then, $q_z < q_w$. Indeed, suppose otherwise that $q_w < q_z$ (recall that all endpoints are assumed to be distinct). Then, since $z \in N_1(u) \cup N_{12}(u)$, there exists a vertex $x \in D_1(u, R)$, i.e. with $T_x \ll_R T_u$, such that $T_z \cap T_x \neq \emptyset$. Since $v, w \in N_0(u) \cup N_2(u)$, it follows that $T_v \cap T_x = \emptyset$ and $T_w \cap T_x = \emptyset$, and thus, $T_x \ll_R T_v$ and $T_x \ll_R T_w$. Therefore, since $p_v < p_z$ and $q_w < q_z$, we obtain that $T_x \ll_R T_z$, and thus, $T_z \cap T_x = \emptyset$, which is a contradiction. It follows that $q_z < q_w$. Moreover, z is adjacent to all vertices x in G, whose trapezoid T_x has an endpoint on L_1 between p_v and p_z, including p_v. Indeed, otherwise, $T_x \ll_R T_z$, and thus, $T_x \ll_R T_u$, since $l(T_z) \ll_R l(T_u)$. This is however a contradiction, since $x \in N(u)$, as we have proved above. Similarly, if $q_w < q_z < q_u$, then $p_z < p_v$ and z is adjacent to all vertices x in G, whose trapezoid T_x has an endpoint on L_2 between q_w and q_z, including q_w.

We construct now from R a new trapezoid representation R' of G as follows. First, for all vertices $z \in N_1(u) \cup N_{12}(u)$ with $l(T_z) \ll_R l(T_u)$, for which $p_v < p_z < p_u$ (and thus $q_z < q_w$), we move the endpoint p_z of $l(T_z)$ directly before p_v on L_1. In the sequel, for all vertices $z' \in N_1(u) \cup N_{12}(u)$ with $l(T_{z'}) \ll_R l(T_u)$, for which $q_w < q_{z'} < q_u$ (and thus $p_z < p_v$), we move the endpoint $q_{z'}$ of $l(T_{z'})$ directly before q_w on L_2. During the movement of all these lines $l(T_z)$ (resp. $l(T_{z'})$), we keep the same relative positions of their endpoints p_z on L_1 (resp. $q_{z'}$ on L_2) as in R, and thus we introduce no new line intersection among the lines of the trapezoids of G. Since all these vertices z (resp. z') are adjacent to all vertices x of G, whose trapezoid T_x has an endpoint on L_1 (resp. L_2) between p_v and p_z, including p_v (resp. between q_w and q_z, including q_w), these movements do not remove any adjacency from, and do not add any new adjacency to G.

Finally, we move both endpoints p_u and q_u of $l(T_u)$ directly before p_v and q_w on L_1 and L_2, respectively. Since u is adjacent to all vertices x, for which T_x has an endpoint between p_v and p_u on L_1, or between q_w and q_u on L_2 in R, the resulting representation R' is a trapezoid representation of G, in which the first condition of Definition 7 is satisfied. Since we moved all lines $l(T_z)$ and $l(T_{z'})$ to the left of T_v and T_w, R' has no additional line intersections than R. Moreover, note that for any line intersection of two lines a and b in R', the relative position of the endpoints of a and b on L_1 and L_2 remains the same as in R. In the case where $p_v > p_u$ (resp. $q_w > q_u$) we replace in the above construction p_v by p_u (resp. q_w by q_u), while in the case where $N_0(u) \cup N_2(u) = \emptyset$, we define $R' = R$. An example of the construction of R' is given in Figure 7. In this example, $v \in N_0(u)$, $w \in N_2(u)$, $z_1, z' \in N_1(u)$ and $z_2 \in N_{12}(u)$.

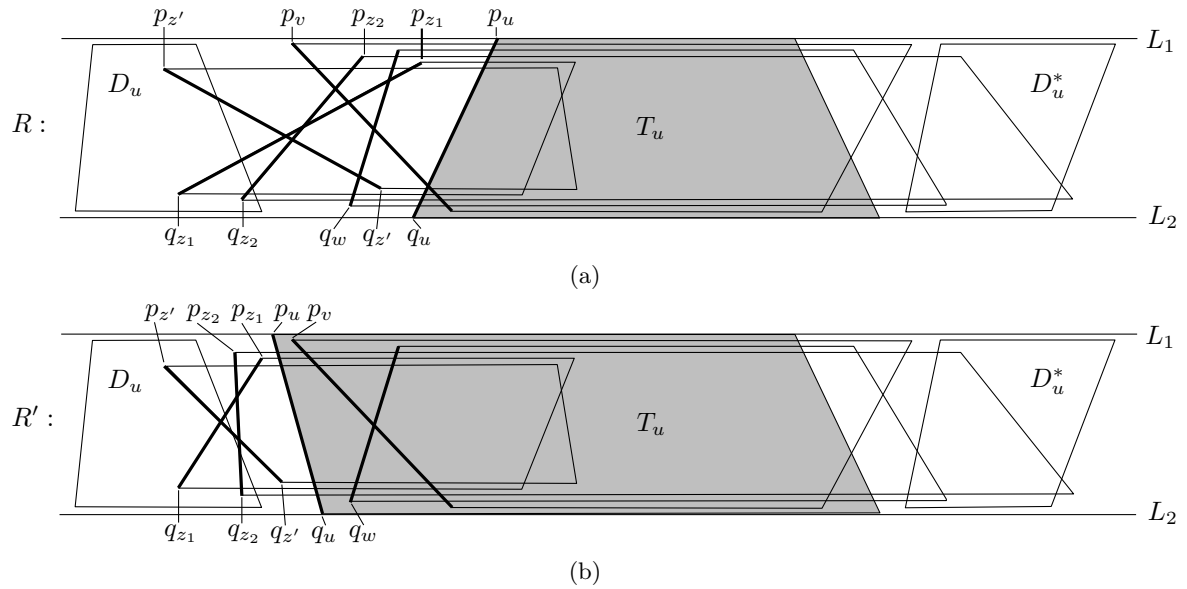

Figure 7: The movement of the left line $l(T_u)$ of the trapezoid T_u, in order to construct a standard trapezoid representation with respect to u.

If R' is not a standard trapezoid representation with respect to u, then we move $r(T_u)$ to the right (similarly to the above), obtaining thus a trapezoid representation R'' of G, in which the second condition of Definition 7 is satisfied. Since during the construction of R'' from R' only the line $r(T_u)$, and other lines that lie completely to the right of $r(T_u)$, are moved to the right, the first condition of Definition 7 is satisfied for R'' as well. Thus, R'' is a standard representation of G with respect to u. Similarly to R', R'' has no additional line intersections than R. Moreover, for any line intersection of two lines a and b in R'', the relative position of the endpoints of a and b on L_1 and L_2 remains the same as in R.

Since R'' is standard with respect to u, the left line $l(T_u)$ of T_u in R'' intersects exactly with those trapezoids T_z, for which $z \in N_1(u) \cup N_{12}(u)$. On the other hand, the right line $r(T_u)$ of T_u in R'' intersects exactly with those trapezoids T_z, for which $z \in N_2(u) \cup N_{12}(u)$. Thus, if we replace in R'' the trapezoid T_u by the two trivial trapezoids (lines) $l(T_u)$ and $r(T_u)$, we obtain a trapezoid representation $R^{\#}(u)$ of the graph $G^{\#}(u)$ defined in Definition 6.

Consider now a vertex $v \in \{u_2, u_3, \ldots, u_k\}$. Due to the assumption, $\delta_v^* \neq \emptyset$ in G, before the vertex splitting of u, and thus, $N_2(v) \cup N_{12}(v) \neq \emptyset$ and $N_1(v) \cup N_{12}(v) \neq \emptyset$ in G by Lemmas 9 and 10. We will prove that $\delta_v^* \neq \emptyset$ in the trapezoid graph $G^{\#}(u)$ as well, after the vertex splitting of u. Due to Lemma 9, it suffices to show that $N_2(v) \cup N_{12}(v) \neq \emptyset$ in $G^{\#}(u)$. Let V_i be the master component of v in G that corresponds to δ_v, before the vertex splitting of u. We may assume w.l.o.g. that $R''(V_i) \ll_{R''} T_v$, by possibly performing a vertical axis flipping of R''. By Lemma 4, $N_1(v) \cup N_{12}(v) = N_1(v, R'') \cup N_{12}(v, R'')$ and $N_2(v) \cup N_{12}(v) = N_2(v, R'') \cup N_{12}(v, R'')$, i.e. these are the sets of neighbors of v in G, whose trapezoids intersect with the trapezoids of $D_1(v, R'')$ and $D_2(v, R'')$ in R'', respectively. Since $N_1(v, R'') \cup N_{12}(v, R'') \neq \emptyset$ and $N_2(v, R'') \cup N_{12}(v, R'') \neq \emptyset$ in G, and since $R^{\#}(u)$ is obtained from R'' by replacing the trapezoid T_u with the lines $l(T_u)$ and $r(T_u)$, it follows easily that $N_1(v, R^{\#}(u)) \cup N_{12}(v, R^{\#}(u)) \neq \emptyset$ and $N_2(v, R^{\#}(u)) \cup N_{12}(v, R^{\#}(u)) \neq \emptyset$ as well. Let V_k be the master component of v in $G^{\#}(u)$ that corresponds to δ_v, after the vertex splitting of u. If V_k lies to the left (resp. right) of T_v in $R^{\#}(u)$, then $N_2(v) \cup N_{12}(v)$ in $G^{\#}(u)$ equals to $N_2(v, R^{\#}(u)) \cup N_{12}(v, R^{\#}(u))$ (resp. to $N_1(v, R^{\#}(u)) \cup N_{12}(v, R^{\#}(u))$, by performing a vertical axis flipping of $R^{\#}(u)$). Therefore, $N_2(v) \cup N_{12}(v) \neq \emptyset$, and thus, $\delta_v^* \neq \emptyset$ in $G^{\#}(u)$, after the vertex splitting of u.

Applying iteratively the above construction for $u = u_i$, $i = 2, 3, \ldots, k$, i.e. by splitting sequentially all vertices of U exactly once, we obtain after k vertex splittings, and after removing from the resulting graph the vertices of $\overline{U} = V(G) \setminus U$, a trapezoid representation $R^{\#}(U)$ of the graph

$G^{\#}(U)$ returned by Algorithm Split-U. Since every trapezoid T_u, $u \in U$, has been replaced by two trivial trapezoids, i.e. lines, in $R^{\#}(U)$, it follows that $G^{\#}(U)$ is a permutation graph with $2k$ vertices, and $R^{\#}(U)$ is a permutation representation of $G^{\#}(U)$.

Finally, suppose that R is an acyclic trapezoid representation of G. According to Definition 2, let P be the permutation graph with $2n$ vertices corresponding to the left and right lines of the trapezoids in R, R_P be the permutation representation of P induced by R, and $\{u_i^1, u_i^2\}$ be the vertices of P that correspond to the same vertex u_i of G, $i = 1, 2, \ldots, n$. Since R is an acyclic trapezoid representation of G, it follows by Definition 2 that R_P is an acyclic permutation representation with respect to $\{u_i^1, u_i^2\}_{i=1}^n$. That is, the simple directed graph F_{R_P} obtained (according to Definition 1) by merging u_i^1 and u_i^2 in P into a single vertex u_i, for every $i = 1, 2, \ldots, n$, has no directed cycle.

Since, during the construction of $R^{\#}(U)$, the trapezoid representation obtained after every vertex splitting has no additional line intersections than the previous one, it follows that $R^{\#}(U)$ has no additional line intersections than R. Moreover, for any line intersection of two lines a and b in $R^{\#}(U)$, the relative position of the endpoints of a and b on L_1 and L_2 remains the same as in R. Thus, the simple directed graph $F_{R^{\#}(U)}$ obtained (according to Definition 1) by merging u_i^1 and u_i^2 in $G^{\#}(U)$ into a single vertex u_i, for every $i = 1, 2, \ldots, k$, is a subdigraph of F_{R_P}. Therefore, since F_{R_P} has no directed cycle, $F_{R^{\#}(U)}$ has no directed cycle as well, i.e. $G^{\#}(U)$ is an acyclic permutation graph with respect to $\{u_i^1, u_i^2\}_{i=1}^k$. This completes the theorem. ■

Appendix E: Proof of Lemma 5

Suppose that P_ϕ is acyclic with respect to $\{u_i^1, u_i^2\}_{i=1}^m$, and let R_0 be an acyclic permutation representation of P_ϕ with respect to $\{u_i^1, u_i^2\}_{i=1}^m$. Then, in particular, R_0 is acyclic with respect to $\{a_i, b_i\}, \{c_i, d_i\}, \{e_i, f_i\}$, for every $i = 1, 2, \ldots, k$. We will construct a truth assignment of the variables $x_1, x_2, \ldots, x_n$ that NAE-satisfies ϕ, as follows. For every $i = 1, 2, \ldots, k$, we define $x_{r_{i,1}} = 1$ if and only if $\theta(c_i) < \theta(a_i)$ in R_0, $x_{r_{i,2}} = 1$ if and only if $\theta(b_i) < \theta(e_i)$ in R_0, and $x_{r_{i,3}} = 1$ if and only if $\theta(f_i) < \theta(d_i)$ in R_0.

Note that this assignment is consistent; that is, all variables $x_{r_{i,j}}$ that correspond to the same x_k are assigned the same value. Indeed, the existence of the lines $u_{i,j}^{i',j'}, v_{i,j}^{i',j'}$ (cf. the bold lines in Figure 5(a)) forces all pairs of crossing lines $\{\ell_{i,j}^1, \ell_{i,j}^2\}$ in the same block to correspond to either 0 or 1 in the assignment.

Now, we show that in each clause α_i, $i = 1, 2, \ldots, k$, there exists at least one true and at least one false variable. For an arbitrary index $i = 1, 2, \ldots, k$, let P_i be the subgraph induced by the vertices $a_i, b_i, c_i, d_i, e_i, f_i$ in P_ϕ, and R_i be the permutation representation of P_i, which is induced by R_0. According to Definition 1, we construct the simple directed graph F_{R_i} by merging into a single vertex each of the pairs $\{a_i, b_i\}$, $\{c_i, d_i\}$ and $\{e_i, f_i\}$ of vertices of P_i. The arc directions of F_{R_i} are implied by the corresponding directions in Φ_{R_i} (or equivalently, in Φ_{R_0}). Then, since R_0 is acyclic with respect to $\{a_i, b_i\} \cup \{c_i, d_i\} \cup \{e_i, f_i\}$, so is R_i. Thus, it follows by Definition 1 that F_{R_i} has no directed cycle. Therefore, the edges c_ia_i, b_ie_i, and f_id_i of P_ϕ have such directions in Φ_{R_0} that it does not hold simultaneously $c_ia_i, b_ie_i, f_id_i \in \Phi_{R_0}$, or $a_ic_i, e_ib_i, d_if_i \in \Phi_{R_0}$. That is, it does not hold simultaneously $\theta(c_i) < \theta(a_i)$, $\theta(b_i) < \theta(e_i)$, and $\theta(f_i) < \theta(d_i)$, or $\theta(a_i) < \theta(c_i)$, $\theta(e_i) < \theta(b_i)$, and $\theta(d_i) < \theta(f_i)$ in R_0, respectively. Then, by the definition of the above truth assignment, it follows that it does not hold simultaneously $x_{r_{i,1}} = x_{r_{i,2}} = x_{r_{i,3}} = 1$, or $x_{r_{i,1}} = x_{r_{i,2}} = x_{r_{i,3}} = 0$, and therefore, the clause $\alpha_i = (x_{r_{i,1}} \vee x_{r_{i,2}} \vee x_{r_{i,3}})$ is NAE-satisfied. Finally, since this holds for every $i = 1, 2, \ldots, k$, ϕ is NAE-satisfiable. ■

Appendix F: Proof of Lemma 6

Let $i \in \{1, 2, \ldots, m\}$. Recall that, by Definition 5, $D_1(u_i, R_H)$ (resp. $D_2(u_i, R_H)$) denotes the set of trapezoids of H_ϕ that lie completely to the left (resp. to the right) of T_{u_i} in R_H. In particular,

$T_{u_i,2}, T_{u_i,3} \in D_1(u_i, R_H)$ and $T_{u_i,5}, T_{u_i,6} \in D_2(u_i, R_H)$. By the construction of R_H, it is easy to see that $T_{u_i,2} \cup T_{u_i,3}$ (resp. $T_{u_i,5} \cup T_{u_i,6}$) is the rightmost (resp. leftmost) connected component of $D_1(u_i, R_H)$ (resp. $D_2(u_i, R_H)$). Thus, $N(V_k) \subseteq N(\{u_{i,2}, u_{i,3}\})$ (resp. $N(V_\ell) \subseteq N(\{u_{i,5}, u_{i,6}\})$), for every connected component V_k (resp. V_ℓ) of $D_1(u_i, R_H)$ (resp. $D_2(u_i, R_H)$). Let V_p be the master component of u_i, such that $D_{u_i} = V_p$. Then, either $V_p = \{u_{i,2}, u_{i,3}\}$, or $V_p = \{u_{i,5}, u_{i,6}\}$. In the case where $V_p = \{u_{i,2}, u_{i,3}\}$, we have $u_{i,4} \in N(\{u_{i,5}, u_{i,6}\}) \not\subseteq N(V_p)$, and thus $\{u_{i,5}, u_{i,6}\} \in \delta^*_{u_i}$. In the case where $V_p = \{u_{i,5}, u_{i,6}\}$, we have $u_{i,1} \in N(\{u_{i,2}, u_{i,3}\}) \not\subseteq N(V_p)$, and thus, $\{u_{i,2}, u_{i,3}\} \in \delta^*_{u_i}$. This proves the lemma. ■

Appendix G: Proof of Theorem 2

Since a graph is a bounded tolerance graph if and only if it is a parallelogram graph [2, 21], it suffices to prove that ϕ is NAE-satisfiable if and only if the trapezoid graph H_ϕ is a parallelogram graph.

($\Leftarrow$) Suppose that H_ϕ is a parallelogram graph, and let $U = \{u_1, u_2, \ldots, u_m\}$. Then, H_ϕ is an acyclic trapezoid graph by Lemma 1. Consider the permutation graph $H^\#_\phi(U)$ with $2m$ vertices, which is obtained by Algorithm Split-U on H_ϕ. Starting with the trapezoid representation R_H of H_ϕ, we obtain by the construction of Theorem 1 (cf. Appendix D) a permutation representation $R^\#_H(U)$ of $H^\#_\phi(U)$. Note that, since R_H is a standard trapezoid representation of H_ϕ with respect to every u_i, $i = 1, 2, \ldots, m$, the line u_i^1 (resp. u_i^2) of T_{u_i} is not moved during the construction of $R^\#_H(U)$ from R_H, for every $i = 1, 2, \ldots, m$. Therefore, $H^\#_\phi(U) = P_\phi$. On the other hand, since by Lemma 6 $\delta^*_{u_i} \neq \emptyset$ for every vertex $u_i \in U$, and since H_ϕ is an acyclic trapezoid graph, Theorem 1 implies that $H^\#_\phi(U) = P_\phi$ is an acyclic permutation graph with respect to $\{u_i^1, u_i^2\}_{i=1}^m$. Thus, ϕ is NAE-satisfiable by Lemma 5.

($\Rightarrow$) Conversely, suppose that ϕ has a NAE-satisfying truth assignment τ. We will construct first a permutation representation R_0 of P_ϕ, and then two trapezoid representations R'_0 and R''_0 of G_ϕ and H_ϕ, respectively, as follows. Similarly to the representation R_P, the representation R_0 has n blocks, i.e. connected components, one for each variable $x_1, x_2, \ldots, x_n$. R_0 is obtained from R_P by performing a horizontal axis flipping of every block, which corresponds to a variable $x_p = 0$ in the truth assignment τ. Every other block, which corresponds to a variable $x_p = 1$ in the assignment τ, remains the same in R_0, as in R_P. Thus, $\theta(\ell^1_{i,j}) > \theta(\ell^2_{i,j})$ if $x_{r_{i,j}} = 1$ in τ, and $\theta(\ell^1_{i,j}) < \theta(\ell^2_{i,j})$ if $x_{r_{i,j}} = 0$ in τ, for every pair $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ of lines in R_0 (which correspond to the literal $x_{r_{i,j}}$ of the clause α_i in ϕ). An example of the construction of this representation R_0 of P_ϕ for the truth assignment $\tau = (1, 1, 0, 0)$ is illustrated in Figure 5(a).

Since τ is a NAE-satisfying truth assignment of ϕ, at least one literal is true and at least one is false in τ in every clause α_i, $i = 1, 2, \ldots, k$. Thus, there are six possible truth assignments for every clause, namely $(1, 1, 0)$, $(1, 0, 1)$, $(0, 1, 1)$, $(0, 0, 1)$, $(0, 1, 0)$, and $(1, 0, 0)$. For the first three ones, we can assign appropriate angles to the lines a_i, b_i, c_i, d_i, e_i, and f_i in the representation R_0, such that the relative positions of all endpoints in L_1 and L_2 remain unchanged, and such that a_i is parallel to b_i, c_i is parallel to d_i, and e_i is parallel to f_i, as illustrated in Figure 8. The last three truth assignments of α_i are the complement of the first three ones. Thus, by performing a horizontal axis flipping of the blocks in Figure 8, to which the lines a_i, b_i, c_i, d_i, e_i, and f_i belong, it is easy to see that for these assignments, we can also assign appropriate angles to these lines in the representation R_0, such that the relative positions of all endpoints in L_1 and L_2 remain unchanged, and such that a_i is parallel to b_i, c_i is parallel to d_i, and e_i is parallel to f_i.

Recall that for every two consecutive pairs $\{\ell^1_{i,j}, \ell^2_{i,j}\}$ and $\{\ell^1_{i',j'}, \ell^2_{i',j'}\}$ of lines in R_P (resp. R_0), which belong to the same block, i.e. where $r_{i,j} = r_{i',j'}$, there are two parallel lines $u^{i',j'}_{i,j}, v^{i',j'}_{i,j}$ that intersect both $\ell^1_{i,j}$ and $\ell^1_{i',j'}$. Thus, after assigning the appropriate angles to the lines $\{\ell^1_{i,j}, \ell^2_{i,j}\}$, $i = 1, 2, \ldots, k$, $j = 1, 2, 3$, we can clearly assign the appropriate angles to the lines $u^{i',j'}_{i,j}, v^{i',j'}_{i,j}$, such that the relative positions of all endpoints in L_1 and L_2 remain unchanged, and such that

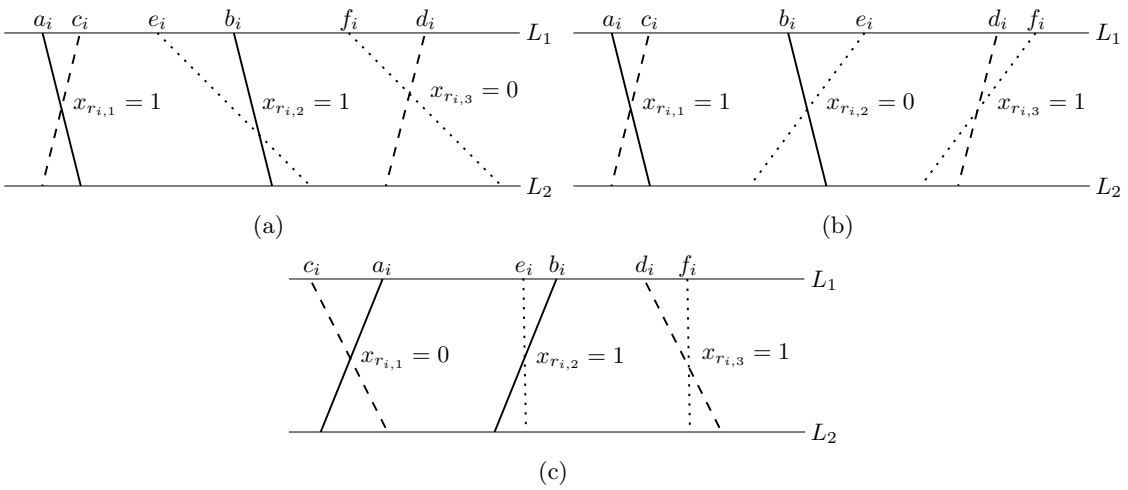

Figure 8: The relative positions of the lines a_i, b_i, c_i, d_i, e_i, and f_i for the truth assignments (a) $(1,1,0)$, (b) $(1,0,1)$, and (c) $(0,1,1)$ of the clause α_i.

$u_{i,j}^{i',j'}$ remains parallel to $v_{i,j}^{i',j'}$. Summarizing, the lines u_i^1 and u_i^2 are parallel in R_0, for every $i = 1, 2, \ldots, m$.

We construct now the trapezoid representation R_0' of G_ϕ from the permutation representation R_0, by replacing for every $i = 1, 2, \ldots, m$ the lines u_i^1 and u_i^2 by the trapezoid T_{u_i}, which has u_i^1 and u_i^2 as its left and right lines, respectively. Since R_0 is obtained by performing horizontal axis flipping of some blocks of R_P, and then changing the angles of the lines, while respecting the relative positions of the endpoints, R_0' is indeed another trapezoid representation of G_ϕ than R_G. Since u_i^1 is now parallel to u_i^2 for every $i = 1, 2, \ldots, m$, it follows clearly that R_0' is a parallelogram representation, and thus, G_ϕ is a parallelogram graph.

Finally, we construct the trapezoid representation R_0'' of H_ϕ from R_0', similarly to the construction of R_H from R_G. Namely, we add for every trapezoid T_{u_i}, $i = 1, 2, \ldots, m$, six parallelograms $T_{u_{i,1}}, T_{u_{i,2}}, \ldots, T_{u_{i,6}}$, resulting in a trapezoid graph with $7m$ vertices. Since in R_0'' the parallelograms $T_{u_{i,1}}$, $T_{u_{i,2}}$, and $T_{u_{i,3}}$ (resp. $T_{u_{i,4}}$, $T_{u_{i,5}}$, and $T_{u_{i,6}}$) are sufficiently close to the left line u_i^1 (resp. right line u_i^2) of T_{u_i}, $i = 1, 2, \ldots, m$, and since between the endpoints of the parallelograms $T_{u_{i,1}}$, $T_{u_{i,2}}$, and $T_{u_{i,3}}$ (resp. $T_{u_{i,4}}$, $T_{u_{i,5}}$, and $T_{u_{i,6}}$) on L_1 and L_2, there are no other endpoints, it follows that R_0'' is indeed another trapezoid representation of H_ϕ than R_H. Finally, since R_0' is a parallelogram representation, and since $T_{u_{i,1}}, T_{u_{i,2}}, \ldots, T_{u_{i,6}}$, $i = 1, 2, \ldots, m$, are all parallelograms, R_0'' is also a parallelogram representation, and thus, H_ϕ is a parallelogram graph. ■

Appendix H: The recognition of tolerance graphs

We show that the reduction from the monotone-NAE-3-SAT problem to the problem of recognizing bounded tolerance graphs presented in Section 3, can be extended to the problem of recognizing general tolerance graphs. In particular, we prove that a given monotone 3-CNF formula ϕ is NAE-satisfiable if and only if the graph H_ϕ constructed in Section 3.2 is a tolerance graph.

Structural properties of tolerance graphs

In the following we assume w.l.o.g. that any tolerance graph has a tolerance representation, in which all tolerances are distinct and no two different intervals share an endpoint [12, 13]. We state now similarly to [13, 14] some definitions and lemmas concerning tolerance graphs. In a certain tolerance representation $\langle I, t\rangle$ of a tolerance graph $G = (V, E)$, a vertex v is called *bounded* if $t_v \leq |I_v|$; otherwise, v is called *unbounded*. An unbounded vertex v of G is called *inevitable* (for a certain tolerance representation), if v is not an isolated vertex, and if setting $t_v = |I_v|$ creates a new edge in the representation, that is, the representation is no longer a tolerance representation

of G. A tolerance representation of G is called *inevitable unbounded*, if every unbounded vertex in this representation is inevitable. For an inevitable unbounded vertex v of G (for a certain tolerance representation), a vertex u is called a *hovering vertex* of v, if $uv \notin E$ and $I_v \subseteq I_u$. The next lemma follows easily from the above definitions.

Lemma 11 *There exists a hovering vertex u for every inevitable unbounded vertex v of the tolerance graph G (for a certain tolerance representation).*

Proof. Since v is an inevitable unbounded vertex, setting $t_v = |I_v|$ creates a new edge in G; let uv be such an edge. Then, clearly $I_u \cap I_v \neq \emptyset$. Since initially $uv \notin E$, it follows that $|I_u \cap I_v| < \min\{t_u, t_v\} \leq t_u$. Furthermore, since setting $t_v = |I_v|$ creates a new edge in G, we obtain that $\min\{t_u, |I_v|\} \leq |I_u \cap I_v| < t_u$, and thus, $|I_u \cap I_v| = |I_v|$, i.e. $I_v \subseteq I_u$. Therefore, since $uv \notin E$ and $I_v \subseteq I_u$, it follows that u is a hovering vertex of v. ■

Lemma 12 ([13]) *Every tolerance representation can be transformed into an inevitable one in $O(n^2)$ time.*

Lemma 13 *Let v be an inevitable unbounded vertex of a tolerance graph G and u be a hovering vertex of v, in a certain tolerance representation of G. Then, $N(v) \subseteq N(u)$ in G.*

Proof. Since v is an inevitable unbounded vertex, $N(v) = \emptyset$. Let $w \in N(v)$ be a neighbor of v in G. Since u is a hovering vertex of v, it follows that $uv \notin E$, and thus, $w \neq u$. Furthermore, since $vw \in E$, and since v is unbounded, we obtain that $\min\{t_v, t_w\} \leq |I_v \cap I_w| \leq |I_v| < t_v$, and thus, $t_w \leq |I_v \cap I_w|$. Then, since $I_v \subseteq I_u$, it follows that $|I_v \cap I_w| \leq |I_u \cap I_w|$, and thus, $t_w \leq |I_u \cap I_w|$, i.e. $w \in N(u)$. Therefore, $N(v) \subseteq N(u)$ in G. ■

The reduction

Consider now a monotone 3-CNF formula ϕ and the trapezoid graph H_ϕ constructed from ϕ in Section 3.2.

Lemma 14 *In the trapezoid graph H_ϕ, there are no two vertices u and v, such that $uv \notin E(H_\phi)$ and $N(v) \subseteq N(u)$ in H_ϕ.*

Proof. The proof is done by investigating all cases for a pair of non-adjacent vertices u, v. First, observe that, by the construction of H_ϕ from G_ϕ, we have $N[u_{i,2}] = N[u_{i,3}]$, $N[u_{i,1}] = N[u_{i,2}] \cup \{u_i\}$, $N[u_{i,5}] = N[u_{i,6}]$, and $N[u_{i,4}] = N[u_{i,5}] \cup \{u_i\}$.

Consider first two vertices u_i and u_k in H_ϕ, for some $i, k = 1, 2, \ldots, m$ and $i \neq k$. Then, by the construction of H_ϕ from G_ϕ, and since u_i and u_k are non-adjacent, $u_{i,1} \in N(u_i) \setminus N(u_k)$ and $u_{k,1} \in N(u_k) \setminus N(u_i)$. Consider next the vertices u_i and $u_{k,j}$, for some $i, k = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, 6$. If $i = k$, then $j \in \{2, 3, 5, 6\}$, since $u_{i,1}, u_{i,4} \in N(u_i)$. In the case where $j \in \{2, 3\}$, we have $u_{i,4} \in N(u_i) \setminus N(u_{k,j})$ and $u_{k,5-j} \in N(u_{k,j}) \setminus N(u_i)$, while in the case where $j \in \{5, 6\}$, we have $u_{i,1} \in N(u_i) \setminus N(u_{k,j})$ and $u_{k,11-j} \in N(u_{k,j}) \setminus N(u_i)$. Suppose that $i \neq k$. Then, it follows by the construction of H_ϕ from G_ϕ that $u_{i,1} \in N(u_i) \setminus N(u_{k,j})$. Furthermore, if $j \in \{1, 2, 3\}$ (resp. $j \in \{4, 5, 6\}$), then $u_{k,j'} \in N(u_{k,j}) \setminus N(u_i)$ for any index $j' \in \{1, 2, 3\} \setminus \{j\}$ (resp. $j' \in \{4, 5, 6\} \setminus \{j\}$).

Consider finally the vertices $u_{i,\ell}$ and $u_{k,j}$, for some $i, k = 1, 2, \ldots, m$ and $\ell, j = 1, 2, \ldots, 6$. If $i = k$, then w.l.o.g. $\ell \in \{1, 2, 3\}$ and $j \in \{4, 5, 6\}$, since $u_{i,\ell}$ and $u_{k,j}$ are non-adjacent. In this case, $u_{i,\ell'} \in N(u_{i,\ell}) \setminus N(u_{k,j})$ and $u_{k,j'} \in N(u_{k,j}) \setminus N(u_{i,\ell})$, for all indices $\ell' \in \{1, 2, 3\} \setminus \{\ell\}$ and $j' \in \{4, 5, 6\} \setminus \{j\}$. Suppose that $i \neq k$. If $j \in \{1, 2, 3\}$ (resp. $j \in \{4, 5, 6\}$), let j' be any index of $\{1, 2, 3\} \setminus \{j\}$ (resp. $\{4, 5, 6\} \setminus \{j\}$). Similarly, if $\ell \in \{1, 2, 3\}$ (resp. $\ell \in \{4, 5, 6\}$), let ℓ' be any index of $\{1, 2, 3\} \setminus \{\ell\}$ (resp. $\{4, 5, 6\} \setminus \{\ell\}$). Then, it follows by the construction of H_ϕ from G_ϕ that $u_{i,\ell'} \in N(u_{i,\ell}) \setminus N(u_{k,j})$ and $u_{k,j'} \in N(u_{k,j}) \setminus N(u_{i,\ell})$.

Therefore, for all possible choices of non-adjacent vertices u, v in the trapezoid graph H_ϕ, we have $N(u) \setminus N(v) \neq \emptyset$ and $N(v) \setminus N(u) \neq \emptyset$, which proves the lemma. ■

Next, we state the proof of Lemma 7.

Proof of Lemma 7. Suppose that H_ϕ is a tolerance graph, and consider a tolerance representation R of H_ϕ. Due to Lemma 12, we may assume w.l.o.g. that R is an inevitable unbounded representation. If R has no unbounded vertices, then we are done. Otherwise, there exists at least one inevitable unbounded vertex v in R, which has a hovering vertex u by Lemma 11, where $uv \notin E(H_\phi)$. Then, $N(v) \subseteq N(u)$ in H_ϕ by Lemma 13, which contradicts Lemma 14. Thus, there exists no unbounded vertex in R, i.e. H_ϕ is a bounded tolerance graph. ■

Theorem 5 *The formula ϕ is NAE-satisfiable if and only if H_ϕ is a tolerance graph.*

Proof. Suppose that ϕ is NAE-satisfiable. Then, by Theorem 2, H_ϕ is a bounded tolerance graph, and thus, H_ϕ is a tolerance graph. Suppose conversely that H_ϕ is a tolerance graph. Then, by Lemma 7, H_ϕ is a bounded tolerance graph. Thus, ϕ is NAE-satisfiable by Theorem 2. ■

Therefore, since monotone-NAE-3-SAT is NP-complete, the problem of recognizing tolerance graphs is NP-hard. Moreover, since the recognition of tolerance graphs lies in NP [17], and since H_ϕ is a trapezoid graph, we obtain Theorem 4.

Appendix D

Miscellaneous

D.1 Edge-simple Circuits Through 10 Ordered Vertices in Square Grids

D.2 On Self-duality of Branchwidth in Graphs of Bounded Genus

D.3 7-[3]coloring Algorithm for Triangle-free Hexagonal Graphs

Edge-Simple Circuits Through 10 Ordered Vertices in Square Grids*

David Coudert[1], Frédéric Giroire[1], and Ignasi Sau[1,2]

[1] Mascotte joint project - INRIA/CNRS/UNSA - Sophia Antipolis, France.
[2] Graph Theory and Combinatorics group - Applied Mathematics IV Department of UPC - Barcelona, Spain.
Firstname.Lastname@sophia.inria.fr

Abstract. A *circuit* in a simple undirected graph $G = (V, E)$ is a sequence of vertices $\{v_1, v_2, \ldots, v_{k+1}\}$ such that $v_1 = v_{k+1}$ and $\{v_i, v_{i+i}\} \in E$ for $i = 1, \ldots, k$. A circuit C is said to be *edge-simple* if no edge of G is used twice in C. In this article we study the following problem: which is the largest integer k such that, given any subset of k ordered vertices of an infinite square grid, there exists an edge-simple circuit visiting the k vertices in the prescribed order? We prove that $k = 10$. To this end, we first provide a counterexample implying that $k < 11$. To show that $k \geq 10$, we introduce a methodology, based on the notion of core graph, to reduce drastically the number of possible vertex configurations, and then we test each one of the resulting configurations with an ILP solver.

Keywords: square grid, edge-simple circuit, prescribed vertices, ILP solver.

1 Introduction

A *circuit* in a simple undirected graph $G = (V, E)$ is a sequence of vertices $\{v_1, v_2, \ldots, v_{k+1}\}$ such that $v_1 = v_{k+1}$ and $\{v_i, v_{i+i}\} \in E$ for $i = 1, \ldots, k$. A circuit C is said to be *edge-simple* if no edge of G is used twice in C. An edge-simple circuit is also called *closed trail* in the literature. The existence of a circuit through a prescribed set of vertices or edges has been an important graph-theoretical question for many years [2–4, 7–9, 11–13, 16, 17, 20]. Typically, high connectivity is a powerful sufficient condition for the existence of such circuits. For instance, it is well known that in a k-vertex-connected graph any subset of k nodes [7] or any subset of $k-1$ independent edges [13] is included in a cycle. A circuit C is a *cycle* if no vertex of G is used twice in C, except for $v_1 = v_{k+1}$.

However, knowing specific properties of the graph often permits to prove much stronger results. In this article we focus on the existence of edge-simple circuits through specified vertices in the infinite square grid (or equivalently, a large enough torus), which is a widely studied 4-connected graph. In addition, we do not require the circuit only to visit a subset of vertices, but also to visit them in a prescribed order. It is

* This work has been partially supported by European project IST FET AEOLUS, PACA region of France, Ministerio de Ciencia e Innovación, European Regional Development Fund under project MTM2008-06620-C03-01/MTM, and Catalan Research Council under project 2005SGR00256.

clear that such a circuit in the square grid always exists for any ordered subset of 4 vertices. After thinking for a few minutes it is also easy to convince oneself that the same holds for 5 vertices. On the other hand, it seems intuitive to suspect that this property will not be true for an arbitrary large subset of ordered vertices of the square grid. Therefore, the following question arises: which is the largest integer k such that, given *any* subset of k ordered vertices of an infinite square grid, there exists an edge-simple circuit visiting the k vertices in the prescribed order? Here, we prove that $k = 10$.

To obtain this result, one has a priori to test the existence of an edge-simple circuit visiting k vertices in the prescribed order on the grid, for all possible placements and orderings of the k vertices. Since the number of possible placements and orderings is prohibitively large, we introduce a methodology, based on the notion of *core graph*, to reduce the number of configurations to be tested. We first provide some background and motivations for the problem in Section 2. We then show in Section 3 that checking the feasibility of a configuration on the grid is equivalent to checking its feasibility on an auxiliary graph, called *internal graph*. Then, in Section 4 we introduce the notion of *core graphs* to reduce drastically the number of internal graphs to be tested. In Section 5 we give a counterexample establishing the upper bound $k \leq 10$. In Section 6 we match this upper bound with an ILP solver to exhaustively test all the orderings for a small list of possible configurations that we obtained after applying the reductions of Sections 3 and 4. Finally, Section 7 concludes the article.

2 Background and Motivation

Connectivity is one of the cornerstone concepts of graph theory. Maybe the most archetypal results are Menger's classical theorems [6], which say that a graph is k-vertex-connected (resp. k-edge-connected) if and only if it contains k vertex-disjoint (resp. edge-disjoint) paths between any two vertices. There is a huge literature concerning extremal problems of cycles in k-connected graphs. For instance, it is well known that in a k-vertex-connected graph any subset of k nodes [7] or any subset of $k-1$ independent edges [13] is included in a cycle. There are a number of works giving necessary or sufficient conditions for the existence of a cycle through a specified set of vertices in a general graph [4, 8, 16, 17].

Some stronger results have been given for specific classes of graphs, like 3-connected cubic graphs [8, 9]. For this class of graphs it is known that there exists a cycle through any 9 vertices, and that there exists a cycle which passes through any 10 given vertices if and only if the graph is not contractible to the Petersen graph [9] in such a way that each of the 10 vertices maps to a distinct vertex of the Petersen graph. If, in addition, the 3-connected cubic graph is planar, then there exists a cycle through any 23 vertices [2]. Another example can be found in [11], where the authors provide necessary and sufficient conditions for a given graph embedded on the torus to contain edge-disjoint cycles satisfying prescribed topological properties.

The disjoint paths problem. Observe that, in a general (di)graph, the problem of deciding whether there exist edge-disjoint paths between given pairs of vertices is NP-complete [15] (even if the graph is a square grid [18]). When the number of pairs of vertices is bounded by a constant, the disjoint paths problem is polynomial in undi-

rected graphs [21], NP-complete in directed graphs [19] (even with only two pairs of vertices [10]), and polynomial in symmetric directed graphs [14].

However, all these results do not take into account the *order* in which the cycle visits the prescribed set of nodes. This is a natural constraint, since for example in telecommunication networks it may be important to connect a subset of nodes in such a way that each node numbered i has capability to communicate only with the two nodes numbered $i-1$ and $i+1$ (modulo the cardinality of the subset of nodes). This could be the case, for instance, of the classical *token ring* networks defined by the standard *IEEE* 802.5. That is, there exists a whole class of problems to consider when the constraint on the order is introduced. In this article we study one of these problems in square grids.

When designing a telecommunication network, the fault tolerance is a crucial issue. Observe that the simplest network which is able to support any single link failure is an edge-simple circuit, and that is one of the main reasons why the study of such circuits is important. The study of the square grid is also natural, due among other reasons to its extensive use in parallel computing. In this context, it is interesting to know which is the largest integer k for which there always exists a circuit visiting any ordered subset of at most k nodes. Observe also that without taking into account the ordering, there exists a cycle (and thus, a circuit) visiting any subset of vertices of the square grid, since the square grid is a Hamiltonian graph.

It is worth mentioning that the square grid is in some sense the common *skeleton* of planar graphs. Indeed it is well-known that every planar graph of branchwidth at least ℓ contains an $(\lfloor \ell/4 \rfloor \times \lfloor \ell/4 \rfloor)$-grid as a minor [22]. Therefore, a square grid is *inside* every planar graph, and any edge-disjoint circuit in a minor of a graph can be easily transformed to an edge-disjoint circuit in the graph itself.

3 Preliminaries

In this section we introduce some definitions to be used throughout. We use standard graph terminology (see, for instance, [6]).

Definition 1 (Configuration, feasible configuration) *A* configuration X *is a subset of vertices of the infinite square grid. A configuration* X *is* feasible *if, for any permutation* σ *of the vertices of* X*, there exists an edge-simple circuit in the infinite square grid joining the vertices of* X *following the ordering given by* σ*.*

Definition 2 (Internal graph, internal and external degree) *Given a subset* $X = \{u_1, \ldots, u_n\}$ *of nodes in the square grid, the* internal graph $G = (V, E)$ *of* X *is the graph with* $V = \{v_1, \ldots, v_n\}$*, and for* $u_i, u_j \in X$*,* $\{v_i, v_j\} \in E$ *if and only if* u_i *and* u_j *are on the same row (or column) and there is no other* $z \in X$ *between* u_i *and* u_j *on that row (or column).*

Given $u \in X$*, the* internal degree $d_{in}(u)$ *of* u *is the degree of* u *in the internal graph* G *of* X*, i.e.,* $d_G(u)$*. Similarly, the* external degree *of* $u \in X$ *is* $d_{out}(u) = 4 - d_{in}(u)$*. A vertex* $u \in X$ *is* isolated *if* $d_{in}(u) = 0$*.*

For example, in Fig. 1, a configuration X in the square grid (defined by the full dots) and its corresponding internal graph G are depicted. The vertex labeled u satisfies $d_{\text{in}}(u) = 3$ and $d_{\text{out}}(u) = 1$.

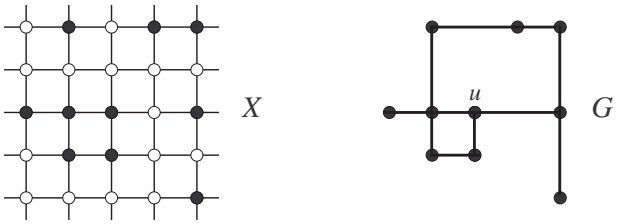

Fig. 1. A configuration X (defined by the full dots) and its corresponding internal graph G.

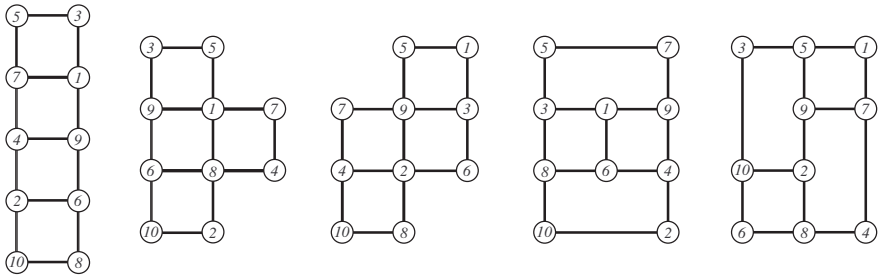

Fig. 2. Some feasible internal graphs on 10 vertices.

Since we deal with an infinite square grid, any two vertices of an internal graph G with external degree at least one can be connected with a path in the grid without using any edge of G. This is because a vertex that has external degree at least one has no neighbor in the internal graph along an infinite semirow or semicolumn of the grid. This fact can be modeled in the following way: given an internal graph G, we construct a (multi)graph $\hat{G}$ from G by adding a new vertex ∞ and, for each vertex $u \in V(G)$, $d_{\text{out}}(u)$ copies of the edge $\{u, \infty\}$.

Definition 3 (Feasible internal graph) *An internal graph G is* feasible *if, for all the permutations σ of the vertices of G, there exists an edge-simple circuit in $\hat{G}$ joining the vertices of G following the ordering given by σ.*

The following lemma follows easily from the above definitions.

Lemma 1. *A configuration X is* feasible *if its internal graph G is feasible.*

Observe that the fact that G is feasible is a sufficient (but *not* necessary) condition for X to be feasible. Intuitively, the internal graph captures the most difficult case among all the configurations having the same internal graph.

Before getting into technical results, and in order to get familiar with the problem, the curious reader may verify that the internal graphs on 10 vertices depicted in Fig. 2 (together with a numbering of their vertices) are feasible. We shall see in Section 6 that this fact is not a coincidence, since any configuration on 10 vertices is feasible.

4 Reducing the Problem

We now prove several technical lemmata to be used in the sequel of the article. The objective is to reduce the number of configurations to be tested.

Lemma 2. *Any internal graph in which all vertices have external degree at least* 2 *is feasible.*

Proof: Let G be an internal graph in which all vertices have external degree at least 2, and assume that the vertices are ordered $v_1, v_2, \ldots, v_k$ by the permutation σ. Then the circuit $\{v_1, \infty, v_2, \infty, v_3, \ldots, v_{k-1}, \infty, v_k, \infty, v_1\}$ is a solution in G. □

Lemma 3. *If an internal graph G is feasible then any internal graph H that is a subgraph of G is feasible.*

Proof: Let G be a feasible internal graph, and let H be a subgraph of G. Assume first that $|V(H)| = |V(G)|$, and let $v_1, \ldots, v_k$ be an ordering of the vertices of H. Consider a solution C in G for the same ordering $v_1, \ldots, v_k$ of the vertices of G. A solution in H is obtained from C by replacing each each $\{u, v\} \in E(G) \setminus E(H)$ with the edges $\{u, \infty\}, \{\infty, v\}$. Otherwise, if $|V(H)| = k < n = |V(G)|$, given an ordering $v_1, \ldots, v_k$ of $V(H)$, consider a solution C in G for an ordering of $V(G)$ that coincides with $v_1, \ldots, v_k$ when restricted to $V(H)$. Then the above replacement transforms C into a solution in H. □

Two internal graphs G_1 and G_2 are said to be *equivalent* if G_2 is feasible if and only if G_1 is.

Lemma 4. *Any two isomorphic internal graphs G_1 and G_2 are equivalent.*

Proof: If G_1 and G_2 are isomorphic, so are $\hat{G_1}$ and $\hat{G_2}$. Therefore, either both or none of them are feasible. □

The proofs of the two following lemmas can be found in [5].

Lemma 5. *If an internal graph G is feasible, then any internal graph G' that can be obtained from G via the following transformation T_1 is also feasible:*

(1) Choose from G an isolated vertex u and an edge $\{x, y\}$.
(2) Remove u, add a new vertex v, and replace the edge $\{x, y\}$ with the edges $\{x, v\}, \{v, y\}$.

Lemma 6. *If an internal graph G is feasible, then any internal graph G' that can be obtained from G via the following transformation T_2 is also feasible:*

(1) Choose from G two vertices u and w, such that u is isolated and $d_{in}(w) \leq 3$.
(2) Remove u, and add a new vertex v and the edge $\{w, v\}$.

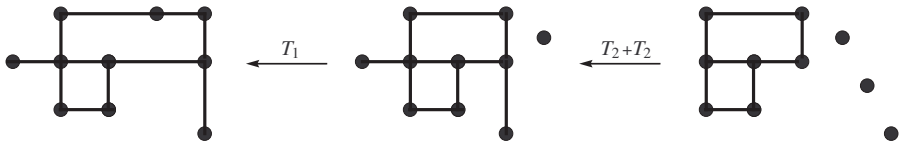

Fig. 3. We can restrict ourselves to core graphs. An arrow from a graph G to a graph H means that if G is feasible, so is H (due to either transformation T_1 or transformation T_2).

Combining inductively Lemmas 5 and 6, we deduce that if G' is an internal graph obtained from a feasible graph G with a sequence of the transformations T_1 and T_2, then G' is also feasible. In practice, this means that in any internal graph we can take the vertices that lie in the *middle* of a path and the vertices with internal degree one, and put them as isolated vertices. If the resulting graph is a feasible internal graph, then by Lemmas 5 and 6, so is the original one. In other words, we can restrict ourselves to internal graphs G whose connected components (except isolated vertices) have at least two vertices in each row and each column.

Definition 4 (Core graph, ℓ-core graph) *An internal graph is a* core graph *if all its non-edgeless connected components have at least two vertices in each row and each column. A core graph G on k vertices is an* ℓ-core graph *if G has $k-\ell$ isolated vertices.*

Lemmas 5 and 6 imply that we can restrict ourselves to core graphs. For instance, consider the example of Fig. 3. The leftmost internal graph (which is the same example of Fig. 1) can be obtained by a sequence of the transformations T_1 and T_2. Thus, to prove that the three internal graphs of Fig. 3 are feasible it is enough to prove it for the rightmost graph, which is a 7-core graph.

This simplification reduces the number of configurations dramatically. In particular, the above discussion together with Lemma 2 proves that all forests are feasible. Therefore, if we want to know if all the configurations on k vertices are feasible, it suffices to test all the core graphs on k vertices; this is the topic of Section 6 for $k = 10$. Summarizing,

Proposition 1. *If all the core graphs on k vertices are feasible, then all the configurations on k vertices are feasible.*

Note that if all the configurations on k vertices are feasible, then clearly so are all the configurations on k' vertices, for every $k' < k$.

We introduce a last criterium to deduce the feasibility of an internal graph on 10 vertices. The proof can be found in [5].

Lemma 7. *All the 10-core graphs on 10 vertices whose non-edgeless connected components can be obtained from a triple edge by subdividing edges are feasible.*

5 Upper Bound: $k < 11$

In this section we show an unfeasible counterexample proving that $k < 11$. For the sake of the presentation, we first describe a simple configuration showing that $k < 12$.

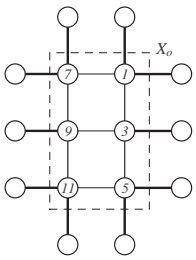

Fig. 4. Counterexample showing that $k < 12$.

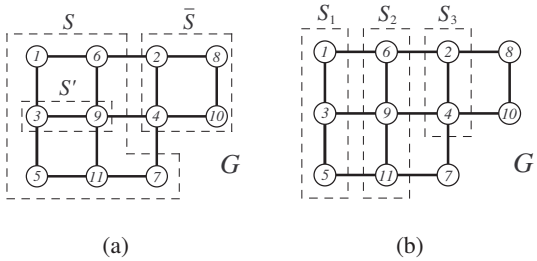

Fig. 5. Counterexample (G, σ) of Proposition 2 showing that $k < 11$, together with the vertex sets defined in the proof.

Given a set $X = \{1, 2, 3, \ldots, 12\}$ of ordered nodes in a square grid G, let X_e (resp. X_o) be the subset of nodes of X with an even (resp. odd) number, and note that any path joining two consecutive vertices must go from X_o to X_e, or viceversa. Let X_o be the set displayed in Fig. 4. Then, regardless of the placement of X_e, we need at least 12 edges outgoing from the graph induced by X_o to route the 12 paths, but there are only 10 such edges (the thick edges in Fig. 4). So, this configuration is unfeasible for any placement of X_e.

Before providing the counterexample showing that $k < 11$, we need the following definition.

Definition 5 (Internal path) *Given an internal graph G, a permutation σ of X, a solution C to the instance (G, σ), and a subset $S \subseteq X$, an* internal path in S *is a subpath P of C linking two consecutive vertices of X according to σ, such that P is a subgraph of $G[S]$.*

Given a subset of vertices $S \subseteq X$, the *paths originating from* S are paths with at least one endpoint in S.

Proposition 2. $k < 11$.

Proof: Let (G, σ) be the internal graph on 11 vertices together with the ordering depicted in Fig. 5.

Suppose for the sake of contradiction that there exists a solution C to the instance (G, σ). Let $S = \{1, 6, 3, 9, 5, 11, 7\} \subseteq X$, and let $\overline{S} = \{2, 8, 4, 10\}$ (see Fig. 5(a)). Note that there are 12 edges outgoing from $G[S]$ to the rest of the grid.

Claim 1 *C contains exactly 1 internal path in S.*

Proof: Suppose first that there is no internal path in S. Therefore, each path originating from S uses at least 2 edges outgoing from $G[S]$. Since $|S| = 7$, there must be 14 edges in C outgoing from $G[S]$ to the rest of the grid, but there are only 12. Therefore, C contains at least 1 internal path in S.

Suppose now that C contains at least 2 internal paths in S. Let $S' = \{3, 9\}$ (see Fig. 5(a)), and note that there are 6 edges outgoing from $G[S']$. Note also that the only possible internal paths in S are $5 \to 6$, $6 \to 7$, and $11 \to 1$, so any internal path in S must cross S'. Therefore, there can be at most 2 such internal paths, and those 2 paths use 4 edges outgoing from $G[S']$. Thus, only $6 - 4 = 2$ outgoing edges from $G[S']$ are left, which are not enough to route the 4 subpaths in C containing the vertices of S'. Therefore, C contains exactly 1 internal path in S. □

Claim 1 implies all the edges outgoing from $G[S]$ are used by C to route paths originating at S. Let $S_1 = \{1, 3, 5\}$ and $S_3 = \{2, 4\}$ (see Fig. 5(b)).

Claim 2 *C contains at least 2 internal paths from S_1 to S_3.*

Proof: Note that subgraph $G[S_3]$ has 6 outgoing edges. Since all the edges outgoing from $G[S]$ are used by C, exactly 3 paths go from S to $\overline{S}$ in C. Clearly, the 4 subpaths in C containing the vertices of S_3 use 4 outgoing edges from $G[S_3]$. Note that all paths from S to S_3 are from S_1.

If there is no path in C from S_1 to S_3, then the 3 paths from S to $\overline{S}$ cross S_3, so no edge outgoing from $G[S_3]$ would be left to route the paths originating from S_3, which is a contradiction.

If there is 1 path in C from S_1 to S_3, then 2 paths from S to $\overline{S}$ cross S_3, so altogether the 3 paths from S to $\overline{S}$ use 5 out of the 6 outgoing edges from $G[S_3]$. However, 3 additional outgoing edges from $G[S_3]$ would be needed to route the 3 remaining paths originating from S_3, which is a contradiction. □

Consider now $S_2 = \{6, 9, 11\}$ (see Fig. 5(b)). The subgraph $G[S_2]$ has 8 outgoing edges, and 6 of them are required in C to route the paths originating at S_2, so only 2 edges outgoing from $G[S_2]$ are still available in C. But, by Claim 2, C contains at least 2 internal paths from S_1 to S_3 (which cross S_2), hence using 4 outgoing edges from $G[S_2]$. The proposition follows. □

6 Lower Bound: $k \geq 10$

To show that $k \geq 10$, one has a priori to test all the configurations with 10 vertices on the grid are feasible. But, the number of such configurations is prohibitively big, as testing a single configuration may take a non-negligible (see discussion below). Hence we introduce a methodology, based on the notion of *core graph* (see the results of Section 4), to reduce the number of configurations to be tested.

A naïve strategy to generate all configurations is to consider all the possibilities of placing 10 points in the square grid. However, we showed in Proposition 1 that we only need to consider *core graphs* with 10 vertices (Definition 4). In addition, these core graphs can be considered modulo isomorphism (Lemma 4). It is clear that the smallest integer i such that an i-core on 10 vertices exists is 4, and in that case the non-edgeless connected component of the 4-core is a 4-cycle. Such a core is always feasible due to Lemma 2, because all the vertices have external degree at least 2. It is also easy to check that, due to the topology of the grid, a 5-core cannot exist. One can also verify

that the only 6-core in which not all vertices have external degree at least 2 is a 2×3-grid. Therefore, it is enough to test this 6-core plus all the ℓ-cores on 10 vertices, for $\ell = 7, 8, 9, 10$. The procedure to generate the core graphs to be tested is detailed in Algorithm 1. The complete code and some examples as well can be found at [1].

Algorithm 1 Test configurations on 10 vertices

for $\ell = 6$ to 10 **do**
  generate a list $\mathcal{T}_\ell$ of all the internal graphs on ℓ vertices in an $(\lfloor \frac{\ell}{2} \rfloor \times \lfloor \frac{\ell}{2} \rfloor)$-grid (modulo translations, symmetries, and compression of empty rows or columns)
  // `INTERNAL DEGREE 1:`
  for all $G \in \mathcal{T}_\ell$ such that G has some vertex of internal degree 1 **do**
    remove G from $\mathcal{T}_\ell$ {*Lemma 6*}
  // `EXTERNAL DEGREE AT LEAST 2:`
  for all $G \in \mathcal{T}_\ell$ such that all the vertices of G have external degree at least 2 **do**
    remove G from $\mathcal{T}_\ell$ {*Lemma 2*}
  // `ISOMORPHIC GRAPHS:`
  partition $\mathcal{T}_\ell$ into classes $\mathcal{G}_1, \ldots, \mathcal{G}_n$ of isomorphic graphs
  for $i = 1$ to n **do**
    if there exists $G \in \mathcal{G}_i$ without at least two vertices per row and column **then**
      remove from $\mathcal{T}_\ell$ all the graphs in $\mathcal{G}_i$ {*Lemma 4 and Proposition 1*}
    else
      remove from $\mathcal{T}_\ell$ all the graphs in $\mathcal{G}_i$ except one {*Lemma 4*}
  // `SUBDIVISION OF TRIPLE EDGE:`
  if $\ell = 10$ **then**
    for all $G \in \mathcal{T}_\ell$ such that G can be obtained from a triple edge by subdividing edges **do**
      remove G from $\mathcal{T}_\ell$ {*Lemma 7*}
  // `SUBGRAPHS:`
  for each pair of graphs $G, H \in \mathcal{T}_\ell$ such that H is a subgraph of G **do**
    remove H from $\mathcal{T}_\ell$ {*Lemma 3*}
  $b_\ell \leftarrow 1$
  for each $G \in \mathcal{T}_\ell$ **do**
    $G' \leftarrow G +$ $(10 - \ell)$ isolated vertices
    for each permutation σ of the vertices of G' **do**
      test if (G', σ) is feasible using an LP solver
      if (G', σ) is *not* feasible **then**
        $b_\ell \leftarrow 0$
if $(b_6 \cdot b_7 \cdot b_8 \cdot b_9 \cdot b_{10}) == 1$ **then**
  $k = 10$
else
  $k < 10$

Proposition 3. *The feasibility of any configuration on 10 vertices follows from Algorithm 1.*

Proof: To prove the correctness of Algorithm 1, we analyze each step sequentially.

Steps 1-2: As by definition, core graphs have at least 2 vertices in each row (resp. column) (besides isolated vertices), the number of rows (resp. column) of an l-core is at

most $\lfloor \frac{\ell}{2} \rfloor$. Hence, it is enough to consider internal graphs fitting into an $\left(\lfloor \frac{\ell}{2} \rfloor \times \lfloor \frac{\ell}{2} \rfloor\right)$-grid, except possibly a set of isolated vertices. Moreover, two internal graphs G_1 and G_2 such that G_2 is obtained from G_1 by a translation or a symmetry are clearly equivalent.

In steps 3-6, the algorithm removes from $\mathcal{T}_\ell$ all the internal graphs with some vertex of internal degree 1. This can be done because, by Lemma 6, it is enough to test the internal graphs in which all vertices, except the isolated ones, have internal degree at least 2.

In steps 7-10, the algorithm removes from $\mathcal{T}_\ell$ all the internal graphs such that all vertices have external degree at least 2, which are feasible due to Lemma 2.

In steps 14-15, the algorithm removes from $\mathcal{T}_\ell$ all the isomorphism classes that have some representant without at least 2 vertices per row and column. This can be done by combining Lemma 4 and Proposition 1.

By Lemma 4, all the graphs in an isomorphism class are equivalent, so it is enough to keep one graphs of each class, as it is done in steps 16-18.

The correctness of steps 20-25 (resp. 26-29) follows directly from Lemma 7 (resp. Lemma 3).

In steps 31-39, the feasibility of each pair (G', σ) is tested using an ILP solver to find a solution of the associated integer multicommodity flow problem, as explained in Section 2 (more details about the implementation are available at [1]).

Finally, by Proposition 1, it is clear that $k = 10$ if and only if all the ℓ-core graphs are feasible, for each $\ell \in \{6, 7, 8, 9, 10\}$. □

Remark 1. In step 12 of Algorithm 1, we partition $\mathcal{T}_\ell$ into isomorphism classes. This step could take a non-negligible time if we just test if each pair of graphs are isomorphic. To deal with this problem, we first carry out a sieve according to the sorted degree sequence of the vertices and the sorted degree sequences of the neighbours of each vertex. That is, if two graphs do not have the same sequence of degrees and degrees of the neighbours of each vertex, we infer directly that these two graphs are not isomorphic. This sieve reduces the computation time considerably.

Remark 2. Observe that, due to Lemma 5, the internal graphs without at least 2 vertices per row and column could have been already removed from T_ℓ after step 2. The reason why we kept those graphs until step 15 is that some graphs that do have at least 2 vertices per row and column are isomorphic to graphs without at least 2 vertices per row and column, so we can also remove them from T_ℓ.

Table 1 summarizes the number of ℓ-cores obtained while running Algorithm 1, for $\ell \in \{6, 7, 8, 9, 10\}$. The numbers given in the first row (initial number of internal graphs) follow from the introduction of internal graphs; without it, we would have a much greater number of configurations to test. Note that the results of Section 4 induce an overall reduction from 4714 to 52 graphs.

Testing the feasibility of core graphs. Recall that for each core graph G on 10 vertices, G is feasible if for any ordering of $V(G)$ there is an edge-simple circuit visiting $V(G)$ in the prescribed order. W.l.o.g. we can assign to one of the vertices of G the number 1 of the permutation (modulo cyclic permutations), and then for each core graph one has to test $9! = 362.880$ possibilities.

Graphs \ ℓ	6	7	8	9	10	Total
Initial number of internal graphs	1	7	53	485	4166	4714
Number of isomorphisms	0	3	42	453	4051	4581
Number of subgraphs	0	0	5	10	58	73
Number of single graphs	0	2	6	22	74	104
Final number of internal graphs	1	2	4	10	35	52

Table 1. Number of ℓ-core graphs on 10 vertices in Algorithm 1. A *single* graph is a graph with a line or column with only one vertex.

For each core graph G and permutation σ, the problem we study can be easily formulated as an integer multicommodity flow problem in a graph with unitary capacity on the edges and so as an integer linear program (ILP). Indeed, the existence of an edge-simple circuit C_σ in a core graphs G is equivalent to the existence of k edge-disjoint paths in G between the pairs of vertices (or *commodities*) $\{\sigma(1), \sigma(2)\}, \ldots, \{\sigma(k-1), \sigma(k)\}, \{\sigma(k), \sigma(1)\}$. Thus, a feasible solution of the ILP implies the existence of an edge simple circuit, and this feasibility can be quickly checked using an ILP solver (for instance, CPLEX).

In average, testing the $9!$ permutations for each internal graph takes around 40 minutes on a PC with an Intel Core 2 Duo CPU 2.33GHz running Fedora 8 (see [1]), so testing the 4714 internal graphs would take around 4 months and a half. Testing the 52 remaining graphs has taken just 35 hours and 37 minutes [1].

Running the ILP solver on the configurations given by Algorithm 1, we obtained that all ℓ-cores are feasible for each $\ell \in \{6, 7, 8, 9, 10\}$. Therefore, combining Propositions 2 and 3 yields that

Theorem 1. *There exists an edge-simple circuit through any set of 10 ordered vertices of an infinite square grid.*

7 Concluding Remarks

In this article we showed that given any subset of 10 ordered vertices of an infinite square grid, there exists an edge-simple circuit visiting the 10 vertices in the prescribed order, and that the number 10 cannot be replaced by 11. To do so, we introduced a methodology to reduce the problem to a small number of configurations, which were then exhaustively tested using an ILP solver. The details about the implementation of our algorithm are available at [1]. Finding a purely combinatorial proof of this result remains open.

Another avenue for further research could be to impose a bound on the size of the grid or torus, namely to consider an $(n_1 \times n_2)$-torus and to find the largest integer $k(n_1, n_2)$ such that given any subset of $k(n_1, n_2)$ ordered vertices in an $(n_1 \times n_2)$-torus, there exists an edge-simple circuit visiting the $k(n_1, n_2)$ vertices in the prescribed order.

Another direction is to consider another graphs instead of the square grid, like triangular and hexagonal grids and, more generally, general planar graphs or graphs of bounded treewidth.

Finally, adding the constraint of the prescribed order to the classical problems concerning the existence of circuits (see related work in Section 2), creates a whole family of new problems to consider.

References

1. `http://www-sop.inria.fr/members/Frederic.Giroire/circuit`.
2. R. Aldred, S. Bau, D. Holton, and B. McKay. Cycles through 23 vertices in 3-connected cubic planar graphs. *Graphs and Combinatorics*, 15:373–376, 1999.
3. S. Bau and D. Holton. Cycles containing 12 vertices in 3-connected cubic graphs. *J. Graph Theory*, 15(4):421–429, 1991.
4. J. Bondy and L. Lovász. Cycles through specified vertices of a graph. *Combinatorica*, 1:117–140, 1981.
5. D. Coudert, F. Giroire, and I. Sau. Circuit visiting 10 ordered vertices in infinite grids. Technical Report RR-6910, INRIA, 2009.
6. R. Diestel. *Graph Theory*. Springer-Verlag, 2005.
7. G. Dirac. In abstrakten Graphen vorhandene vollständige 4-Graphen u nd ihre Unterteilungen. *Math. Nachr.*, 22:61–85, 1960.
8. Y. Egawa, R. Glas, and S. Locke. Cycles and paths through specified vertices in k-connected graphs. *J. Comb. Theory Ser. B*, 52:20–29, 1991.
9. M. Ellingham, D. Holton, and C. Little. Cycles through ten vertices in 3-connected cubic graphs. *Combinatorica*, 4:265–273, 1984.
10. S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Journal of Theoretical Computer Science*, 10(2):111–121, 1980.
11. A. Frank and A. Schrijver. Edge-Disjoint Circuits in Graphs on the Torus. *J. Comb. Theory Ser. B*, 55(1):9–17, 1992.
12. F. Göringa, J. Harant, E. Hexel, and Z. Tuzac. On short cycles through prescribed vertices of a graph. *Discrete Mathematics*, 286(1-2):67–74, 2004.
13. R. Häggkvist and C. Thomassen. Circuits through specified edges. *Discrete Math.*, 41:29–34, 1982.
14. A. Jarry and S. Pérennes. Disjoint paths in symmetric digraphs. *Discrete Applied Mathematics*, 157(1):90–97, 2009.
15. R. M. Karp. On the complexity of combinatorial problems. *Networks*, 5:45–68, 1975.
16. K. Kawarabayashi. Cycles through a prescribed vertex set in n-connected graphs. *J. Comb. Theory Ser. B*, 90(2):315–323, 2004.
17. A. Kelmans and M. Lomonosov. When m vertices in a k-connected graph cannot be walked round along a simple cycle. *Discrete Math.*, 38:317–322, 1982.
18. M. Kramer and J. van Leeuwen. Wire-routing is NP-complete. Technical Report RUU-CS-82-4, Department of Computer Science, University of Utrecht, 1982.
19. A. S. LaPaugh and R. L. Rivest. The subgraph homeomorphism problem. *J. Comput. Syst. Sci.*, 20(2):133–149, 1980.
20. L. Lovász. On some connectivity properties of Eulerian graphs. *Acta Math. Acad. Sci. Hungar.*, 28:129–138, 1976.
21. N. Robertson and P. Seymour. Graph minors XIII: the disjoint paths problem. *J. Comb. Theory Ser. B*, 63:65–110, 1995.
22. N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *J. Comb. Theory Ser. B*, 62(2):323–348, 1994.

On Self-Duality of Branchwidth in Graphs of Bounded Genus ⋆

Ignasi Sau [a]

[a] *Mascotte project INRIA/CNRS/UNSA, Sophia-Antipolis, France; and Graph Theory and Comb. Group, Applied Maths. IV Dept. of UPC, Barcelona, Spain.*

Dimitrios M. Thilikos [b]

[b] *Department of Mathematics, National Kapodistrian University of Athens, Greece.*

Abstract

A graph parameter is *self-dual* in some class of graphs embeddable in some surface if its value does not change in the dual graph more than a constant factor. Self-duality has been examined for several width-parameters, such as branchwidth, pathwidth, and treewidth. In this paper, we give a direct proof of the self-duality of branchwidth in graphs embedded in some surface. In this direction, we prove that $\mathbf{bw}(G^*) \leq 6 \cdot \mathbf{bw}(G) + 2g - 4$ for any graph G embedded in a surface of Euler genus g.

Key words: graphs on surfaces, branchwidth, duality, polyhedral embedding.

1 Preliminaries

A *surface* is a connected compact 2-manifold without boundaries. A surface Σ can be obtained, up to homeomorphism, by adding $\mathbf{eg}(\Sigma)$ *crosscaps* to the sphere. $\mathbf{eg}(\Sigma)$ is called the *Euler genus* of Σ. We denote by (G, Σ) a graph G embedded in a surface Σ. A subset of Σ meeting the drawing only at vertices of G is called *G-normal*. If an O-arc is G-normal, then we call it a *noose*. The *length* of a noose is the number of its vertices. *Representativity*, or *face-width*, is a parameter that quantifies local planarity and density of embeddings. The representativity $\mathbf{rep}(G, \Sigma)$ of a graph embedding (G, Σ) is the smallest length of a non-contractible noose in Σ. We call an embedding (G, Σ) *polyhedral* if G is 3-connected and $\mathbf{rep}(G, \Sigma) \geq 3$. See [7] for more details. For a given embedding (G, Σ), we denote by (G^*, Σ) its dual embedding. Thus G^* is the

⋆ This work has been supported by IST FET AEOLUS, COST 295-DYNAMO, and by the Project "Kapodistrias" (ΑΠ 02839/28.07.2008) of the National and Kapodistrian University of Athens (project code: 70/4/8757).

geometric dual of G. Each vertex v (resp. face r) in (G, Σ) corresponds to some face v^* (resp. vertex r^*) in (G^*, Σ). Also, given a set $X \subseteq E(G)$, we denote as X^* the set of the duals of the edges in X.

Given a graph G and a set $X \subseteq E(G)$, we define $\partial X = (\bigcup_{e \in X} e) \cap (\bigcup_{e \in E(G) \setminus X} e)$ (notice that $\partial X = \partial(E(G) \setminus X)$). A *branch decomposition* (T, μ) of a graph G consists of an unrooted ternary tree T (i.e., all internal vertices are of degree three) and a bijection $\mu : L \to E(G)$ from the set L of leaves of T to the edge set of G. For every edge $f = \{t_1, t_2\}$ of T we define the *middle set* $\mathbf{mid}(e) \subseteq V(G)$ as follows: Let L_1 be the leaves of the connected component of $T \setminus \{e\}$ that contain t_1. Then $\mathbf{mid}(e) = \partial\mu(L_1)$. The *width* of (T, μ) is defined as $\max\{|\mathbf{mid}(e)| : e \in T\}$. An optimal branch decomposition of G is defined by a tree T and a bijection μ which give the minimum width, called the *branchwidth* of G, and denoted by $\mathbf{bw}(G)$.

Suppose G_1 and G_2 are graphs with disjoint vertex-sets and $k \geq 0$ is an integer. For $i = 1, 2$, let $W_i \subseteq V(G_i)$ form a clique of size k and let G_i' $(i = 1, 2)$ be obtained from G_i by deleting some (possibly none) of the edges from $G_i[W_i]$ with both endpoints in W_i. Consider a bijection $h : W_1 \to W_2$. We define a *clique-sum* $G_1 \oplus G_2$ of G_1 and G_2 to be the graph obtained from the union of G_1' and G_2' by identifying w with $h(w)$ for all $w \in W_1$.

Let $\mathcal{G}$ be a class of graphs embeddable in a surface Σ. We say that a graph parameter $\mathbf{p}$ is (c, d)-*self-dual* on $\mathcal{G}$ if for every graph $G \in \mathcal{G}$ and for its geometric dual G^*, $\mathbf{p}(G^*) \leq c \cdot \mathbf{p}(G) + d$. Results concerning self-duality of pathwidth can be found in [1, 4]. Branchwidth is $(1, 0)$-self-dual in planar graphs that are not forests [9], while analogous results have been proven for other parameters such as pathwidth [1,3] and treewidth [2,5,6]. In this note, we give a proof that branchwidth is $(6, 2g - 4)$-self-dual in graphs of Euler genus at most g. We also believe that our result can be considerably improved. In particular, we conjecture that branchwidth is $(1, g)$-self-dual.

2 Self-duality of branchwidth

If (G, Σ) is a polyhedral embedding, then the following proposition follows by an easy modification of the proof of [4, Theorem 1].

Proposition 1 *Let (G, Σ) and (G^*, Σ) be dual polyhedral embeddings in a surface of Euler genus g. Then $\mathbf{bw}(G^*) \leq 6 \cdot \mathbf{bw}(G) + 2g - 4$.*

In the sequel, we focus on generalizing Proposition 1 to arbitrary embeddings. For this we first need some technical lemmata, whose proofs are easy or well known, and omitted in this extended abstract. Note that the removal of a vertex in G corresponds to the contraction of a face in G^*, and viceversa.

Lemma 1 *The removal of a vertex or the contraction of a face from an embedded graph decreases its branchwidth by at most 1.*

Lemma 2 (Fomin and Thilikos [3]) *Let G_1 and G_2 be graphs with one edge or one vertex in common. Then* $\mathbf{bw}(G_1 \cup G_2) \leq \max\{\mathbf{bw}(G_1), \mathbf{bw}(G_2), 2\}$.

Theorem 1 *Let (G, Σ) be an embedding with $g = \mathbf{eg}(\Sigma)$. Then* $\mathbf{bw}(G^*) \leq 6 \cdot \mathbf{bw}(G) + 2g - 4$.

Proof. The proof uses the following procedure that applies a series of cutting operations to decompose G into polyhedral pieces plus a set of vertices whose size is linearly bounded by $\mathbf{eg}(\Sigma)$. The input is the graph G and its dual G^* embedded in Σ.

1. Set $\mathcal{B} = \{G\}$, and $\mathcal{B}^* = \{G^*\}$ (we call the members of $\mathcal{B}$ and $\mathcal{B}^*$ *blocks*).
2. If (G, Σ) has a minimal separator S with $|S| \leq 2$, let $C_1, \ldots, C_\rho$ be the connected components of $G[V(G) \setminus S]$ and, for $i = 1, \ldots, \rho$, let G_i be the graph obtained by $G[V(C_i) \cup S]$ by adding an edge with both endpoints in S in the case where $|S| = 2$ and such an edge does not already exist (we refer to this operation as *cutting* G along the separator S). Notice that a (non-empty) separator S of size at most 2 corresponds to a non-empty separator S^* of G^*, and let $G_i^*, i = 1, \ldots, \rho$ be the graphs obtained by cutting G^* along S^*. We say that each G_i (resp G_i^*) is a *block* of G (resp. G^*) and notice that each G and G^* is the clique sum of its blocks. Therefore, from Lemma 2, $\mathbf{bw}(G^*) \leq \max\{2, \max\{\mathbf{bw}(G_i^*) \mid i = 1, \ldots, \rho\}\}$ **(1)**. Observe that we may assume that for each $i = 1, \ldots, \rho$, G_i and G_i^* are embedded in a surface Σ_i such that G_i is the dual of G_i^* and $\mathbf{eg}(\Sigma) = \sum_{i=1,\ldots,\rho} \mathbf{eg}(\Sigma_i)$. Notice that $\mathbf{bw}(G_i) \leq \mathbf{bw}(G), i = 1, \ldots, \rho$ **(2)**, as the possible edge addition does not increase the branchwidth, since each block of G is a minor of G. We set $\mathcal{B} \leftarrow \mathcal{B} \setminus \{G\} \cup \{G_1, \ldots, G_\rho\}$ and $\mathcal{B}^* \leftarrow \mathcal{B}^* \setminus \{G^*\} \cup \{G_1^*, \ldots, G_\rho^*\}$.
3. If (G, Σ) has a non-contractible and non-surface-separating noose meeting a set S with $|S| \leq 2$, let $G' = G[V(G) \setminus S]$ and let F be the set of of faces in G^* corresponding to the vertices in S. Observe that the obtained graph G' has an embedding to some surface Σ' of Euler genus *strictly* smaller than Σ that, in turn, has some dual G'^* in Σ'. Therefore $\mathbf{eg}(\Sigma') < \mathbf{eg}(\Sigma)$. Moreover, G'^* is the result of the contraction in G^* of the $|S|$ faces in F. From Lemma 1, $\mathbf{bw}(G^*) \leq \mathbf{bw}(G'^*) + |S|$ **(3)**. Set $\mathcal{B} \leftarrow \mathcal{B} \setminus \{G\} \cup \{G'\}$ and $\mathcal{B}^* \leftarrow \mathcal{B}^* \setminus \{G^*\} \cup \{G'^*\}$.
4. Apply (recursively) Steps 2–4 for each block $G \in \mathcal{B}$ and its dual.

We now claim that before each recursive call of Steps 2–4, it holds that $\mathbf{bw}(G^*) \leq 6 \cdot \mathbf{bw}(G) + 2\mathbf{eg}(\Sigma) - 4$. The proof uses descending induction on the the distance from the root of the recursion tree of the above procedure. Notice that all embeddings of graphs in the collections $\mathcal{B}$ and $\mathcal{B}^*$ constructed by the above algorithm are polyhedral, except from the trivial case that they are just cliques of size 2. Then the theorem follows directly from Proposition 1.

Suppose that G (resp. G^*) is the clique sum of its blocks $G_1, \ldots, G_\rho$ (resp. $G_1^*, \ldots, G_\rho^*$) embedded in the surfaces $\Sigma_1, \ldots, \Sigma_\rho$ (Step 2). By induction, we have that $\mathbf{bw}(G_i^*) \leq 6 \cdot \mathbf{bw}(G_i) + 2\mathbf{eg}(\Sigma_i) - 4, i = 1, \ldots, \rho$ and the claim follows from Relations **(1)** and **(2)** and the fact that $\mathbf{eg}(\Sigma) = \sum_{i=1,\ldots,\rho} \mathbf{eg}(\Sigma)$.

Suppose now (Step 3) that G (resp. G^*) occurs from some graph G' (resp. G'^*) embedded in a surface Σ' where $\mathbf{eg}(\Sigma') < \mathbf{eg}(\Sigma)$ after adding the vertices in S (resp. S^*). From the induction hypothesis, $\mathbf{bw}(G'^*) \leq 6 \cdot \mathbf{bw}(G') + 2\mathbf{eg}(\Sigma') - 4 \leq 6 \cdot \mathbf{bw}(G') + 2\mathbf{eg}(\Sigma) - 2 - 4$ and the claim follows easily from Relation **(3)** as $|S| \leq 2$ and $\mathbf{bw}(G') \leq \mathbf{bw}(G)$. ■

3 Recent results and a conjecture

Very recently Mazoit [6] proved that treewidth is a $(1, g+1)$-self-dual parameter in graphs embeddable in surfaces of Euler genus g. Using that the branchwidth and the treewidth of a graph G, with $|E(G)| \geq 3$, satisfy $\mathbf{bw}(G) \leq \mathbf{tw}(G)+1 \leq \frac{3}{2}\mathbf{bw}(G)$ [8], this implies that $\mathbf{bw}(G^*) \leq \frac{3}{2}\mathbf{bw}(G)+g+2$, improving the constants of Theorem 1. We believe that an even tighter self-duality relation holds for branchwidth and hope that the approach of this paper will be helpful to settle the following conjecture.

Conjecture 1 *If G is a graph embedded in some surface Σ, then* $\mathbf{bw}(G^*) \leq \mathbf{bw}(G) + \mathbf{eg}(\Sigma)$.

References

[1] O. Amini, F. Huc, and S. Pérennes. On the Pathwidth of Planar Graphs. *SIAM Journal on Discrete Mathematics*, 2009. To appear.

[2] V. Bouchitté, F. Mazoit, and I. Todinca. Chordal embeddings of planar graphs. *Discrete Mathematics*, 273(1-3):85–102, 2003. EuroComb'01 (Barcelona).

[3] F. V. Fomin and D. M. Thilikos. Dominating Sets in Planar Graphs: Branch-Width and Exponential Speed-Up. *SIAM Journal on Computing*, 36(2):281–309, 2006.

[4] F. V. Fomin and D. M. Thilikos. On self duality of pathwidth in polyhedral graph embeddings. *Journal of Graph Theory*, 55(1):42–54, 2007.

[5] D. Lapoire. Treewidth and duality for planar hypergraphs, 1996 (manuscript). `http://www.labri.fr/perso/lapoire/papers/dual_planar_treewidth.ps`.

[6] F. Mazoit. Tree-width of graphs and surface duality. To appear in *DIMAP workshop on Algorithmic Graph Theory (AGT), Warwick, U.K.*, March 2009.

[7] B. Mohar and C. Thomassen. *Graphs on surfaces*. John Hopkins University Press, 2001.

[8] N. Robertson and P. Seymour. Graph minors. X. Obstructions to Tree-decomposition. *J. Comb. Theory Series B*, 52(2):153–190, 1991.

[9] P. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

A 7/6-approximation Algorithm for Multicoloring Triangle-free Hexagonal Graphs

Petra Šparl
University of Maribor
FG, Smetanova 17
2000 Maribor, SLOVENIA
petra.sparl@uni-mb.si

Ignasi Sau
Mascotte project
I3S(CNRS-UNS) and INRIA
F-06902 Sophia-Antipolis, FRANCE
ignasi.sau@sophia.inria.fr

Janez Žerovnik
University of Ljubljana
FS, Aškerčeva 6
1000 Ljubljana, SLOVENIA
janez.zerovnik@imfm.uni-lj.si

Abstract

Given a graph G and a demand function $p : V(G) \to \mathbb{N}$, a proper n-$[p]$*coloring* is a mapping $f : V(G) \to \{1, \ldots, n\}$ such that $|f(v)| \geq p(v)$ for any vertex $v \in V(G)$ and $f(v) \cap f(u) = \emptyset$ for any pair of adjacent vertices u and v. The least integer n for which a proper n-$[p]$coloring exists, $\chi(G)$, is called the *multichromatic number* of G. Finding the multichromatic number of induced subgraphs of the triangular lattice (called *hexagonal graphs*) has important applications in cellular networks. The *weighted clique number* of a graph G, $\omega(G)$, is the maximum weight of a clique in G, where the *weight* of a clique is the total demand of its vertices. McDiarmid and Reed [6] conjectured that $\chi(G) \leq (9/8)\omega(G) + o(1)$ for triangle-free hexagonal graphs. In this article we provide an algorithm to find a 7-[3]coloring of triangle-free hexagonal graphs, which implies that $\chi(G) \leq (7/6)\omega(G) + o(1)$. Our result constitutes a much shorter alternative to the inductive proof of Havet [3] and improves the short proof of Sudeep and Vishwanathan [10], who proved the existence of a 14-[6]coloring.

Keywords: graph algorithm, approximation algorithm, graph coloring, frequency planning, cellular networks.

1 Introduction

A fundamental problem concerning cellular networks is to assign sets of frequencies (colors) to transmitters (vertices) in order to avoid unacceptable interferences [2]. The number of frequencies demanded at a transmitter may vary between transmitters. In a usual cellular model, transmitters are centers of hexagonal cells and the corresponding adjacency graph is a subgraph of the infinite triangular lattice. An integer $p(v)$ is assigned to each vertex of the triangular lattice and will be called the *demand* of the vertex v. The vertex weighted graph induced on the subset of the triangular lattice of vertices of positive demand is called a *hexagonal graph*, and is denoted $G(V, E, p)$. Hexagonal graphs arise naturally in studies of cellular networks. A *proper multicoloring* of G is a mapping f from $V(G)$ to subsets of integers such that $|f(v)| \geq p(v)$ for any vertex $v \in V(G)$ and $f(v) \cap f(u) = \emptyset$ for any pair of adjacent vertices u and v in the graph G. The minimal cardinality of a proper multicoloring of G, $\chi(G)$, is called the *multichromatic number*. Another invariant of interest in this context is the *(weighted) clique number*, $\omega(G)$, defined as follows: The weight of a clique of G is the sum of demands on its vertices and $\omega(G)$ is the maximal clique weight on G. Clearly, $\chi(G) \geq \omega(G)$. Recently, the bound $\chi(G) \leq (4/3)\omega(G) + o(1)$ was independently proved by several authors [6–8, 12].

Better bounds can be obtained for triangle-free hexagonal graphs. The conjecuture due to McDiarmid and Reed [6] is that $\chi(G) \leq (9/8)\omega(G)+o(1)$ holds for triangle-free hexagonal graphs. In [4] a distributed algorithm with competitive ratio 5/4 is given. In [13] the authors report the existence of 2-local distributed algorithm with competitive ratio 5/4, while an inductive proof for ratio 7/6 is reported in [3]. A 2-local 7/6-competitive algorithm for a sub-class of triangle-free hexagonal graphs is given in [14]. An elegant idea that implies the existence of a 14-[6]coloring is presented in [10]. However, it is unclear how to design an algorithm for 7-[3]coloring from these proofs.

In this paper we give an algorithm for 7-[3]coloring an arbitrary triangle-free hexagonal graph G. This provides a much shorter alternative proof to the inductive proof of Havet [3] and improves the short proof of [10] that implied the existence of a 14-[6]coloring.

The paper is organized as follows. In the next section we formally define some basic terminology. In subsection 3.1 we present an algorithm for 7-[3]coloring an arbitrary triangle-free hexagonal graph G. The correctness of the algorithm is proved in subsection 3.2. Section 4 concludes the article.

2 Preliminaries

A *weight function* on a graph G is a function from $V(G)$, the set of vertices of G, into the set of non-negative integers. Let p be a weight function on a graph G. A proper n-$[p]$*coloring* of G is a mapping f from $V(G)$ into the set of subsets of $\{1, 2, \ldots, n\}$, such that $|f(v)| = p(v)$ for every vertex $v \in V(G)$ and such that for any two adjacent vertices u and v of G, $f(u) \cap f(v) = \emptyset$. A special case is when p is a constant function. For example, a 7-[3]coloring is an assignment of colors between 1 and 7 to each vertex.

Following [6], the vertices of the triangular lattice can be represented as integer linear combinations $x\vec{p} + y\vec{q}$ of the two vectors $\vec{p} = (1, 0)$ and $\vec{q} = (\frac{1}{2}, \frac{\sqrt{3}}{2})$ of $\mathbb{R}^2$. Thus, we may identify vertices of triangular grid with pairs (x, y) of integers. Two vertices are adjacent when the Euclidean distance between them is one. Therefore each vertex (x, y) has six neighbors $(x\pm1, y)$, $(x, y \pm 1)$, $(x + 1, y - 1)$, and $(x - 1, y + 1)$. For simplicity, we will refer to the neighbors as R (right), L (left), UR (up-right), DL (down-left), DR (down-right), and UL (up-left), respectively, see Figure 1(a).

There is a natural 3-coloring of the vertices of the infinite triangular lattice, which gives rise to the partition of the vertex set of any hexagonal graph into three independent sets *Red*, *Blue*, and *Green*. According to this partition, each vertex $v \in V(G)$ has its *base color,* namely *red* (R)*, blue* (B), or *green* (G), which is denoted by $c(v)$. Formally, we define

$$c(v) = (x + 2y) \bmod 3 + 1.$$

To avoid confusion, we define the constants $R = -2$, $B = -1$, $G = 0$, and will use R, B, G when referring to the colors of the 7-coloring, see Figure 1(b). We denote this 3-coloring by RBG-coloring.

In addition, we will assign to each vertex $v \in V(G)$ its *base number* defined by

$$n(v) = x \bmod 2 + 2(y \bmod 2) + 1.$$

It is not difficult to see that on each straight line of the triangular lattice there are only two different base numbers which alternate, see Figure 1(c).

Definition 1 *Vertex $v \in V(G)$ is a* right center *if it has at least two of its R, DL, and UL neighbors in G. Similarly, vertex $v \in G$ is a* left center *if it has at least two of its L, DR, and UR neighbors in G.*

Note that in a triangle-fee hexagonal graph, two centers of the same type (left or right) cannot be adjacent. Note also that each cycle contains centers of both types.

Definition 2 *Let c stand for red, blue, or green. A vertex v is* c-free *if its base color and the base colors of its neighbors are all different from c.*

By the last two definitions and because we assume there are no triangles, it is obvious that a center of a triangle-free graph is a c-free vertex. For example, a red right center and a blue left center are both green-free.

Note that all neighbors of a center have the same base color and pairwise different base numbers. Exactly 2 out of the 3 potential neighbors of a center have the same base number parity. Furthermore, the L and RD (resp. R and UL) neighbors of a left (resp. right) center have the same base number parity.

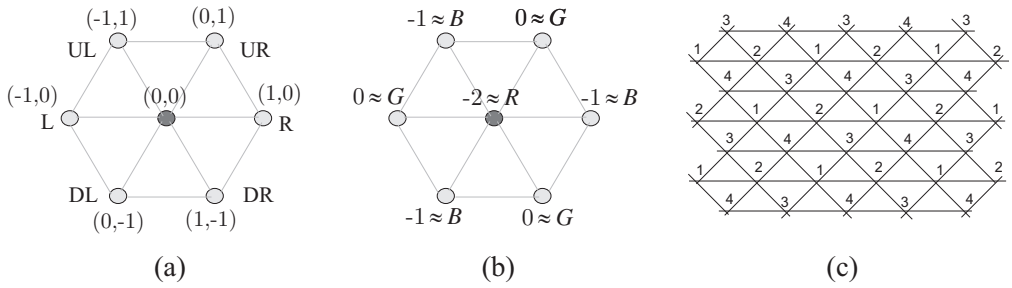

Figure 1: Coordinates, base colors, and base numbers.

Definition 3 *A path $(x_1, x_2, ..., x_n)$ is called* left *(resp.* rightup, rightdown*) if x_{i+1} is the left (resp. rightup, rightdown) neighbor of x_i for $i = 1, 2, ..., n-1$. A* tristar *is the union of one left, one rightup, and one rightdown path emerging from a common origin, which is a left center. Paths of a tristar will be called* rays*.*

Let u, v, and z be the end-vertices of the left, rightup, and rightdown rays of a tristar with left center x. Note that there are only two possibilities for the parities of lengths of the paths $(u, ..., x, ..., v)$, $(u, ..., x, ..., z)$, and $(v, ..., x, ..., z)$. Indeed, either the lengths of all three paths are even (all three rays have the same length parity) or the lengths of two paths are odd and the length of the remaining path is even (rays have different length parity), see Figure 2.

Furthermore, in a tristar with all rays of odd length, the end-vertices and the center have pairwise different base numbers. This easily follows from the fact that on each straight line only two base numbers alternate

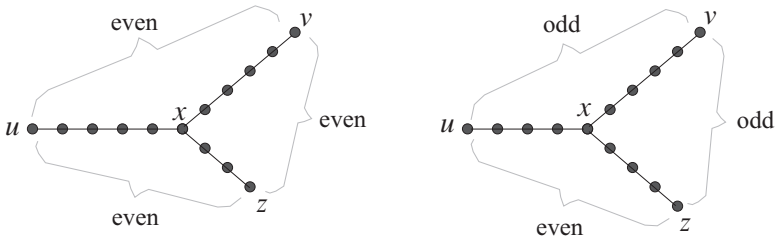

Figure 2: A tristar with all three paths of even distance and a tristar with only one path of even distance.

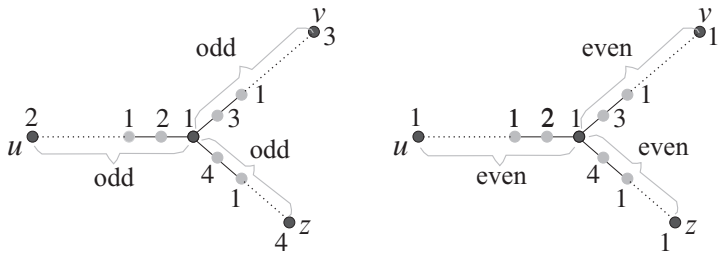

Figure 3: Base numbers of end-vertices of a tristar.

and that the L, RU, and RD neighbors of the central vertex have pairwise different base numbers. Furthermore, the left end-vertex and the rightdown end-vertex have the same base number parity. Similarly, in a tristar with all rays of even length, all end-vertices and the center have the same base number, see Figure 3.

3 7-[3]coloring of a triangle-free hexagonal graph

We describe in this section the algorithm which 7[3]-colors every triangle-free hexagonal graph G. The algorithm uses seven colors, more precisely, the three base colors R, B, and G and four additional colors A, B, C, and D.

Recall that $c(v)$ and $n(v)$ denote the base color and the base number of vertex $v \in V(G)$, respectively. Recall that vertices of each straight line of the triangular lattice have only two different base numbers which alternate.

3.1 The 7-[3]coloring algorithm

The main idea of the algorithm is composed of the following four steps:

- assignment of the base RBG-coloring to the whole graph G;
- partial coloring of some centers;
- creation of an auxiliary graph, which is planar and thus 4-colorable [9];
- extension of the obtained auxiliary graph coloring to the vertices with higher demand in such a way that the final 7-[3]coloring is proper.

We describe now the details of the algorithm. We first need to introduce a definition. A center is called *suitable* if it is either a right center or a left center not incident to a right center.

The algorithm:
Input: a triangle-free hexagonal graph $G = (V, E, p)$, where $p(v) = 3$ for every vertex $v \in V(G)$, base color $c(v)$, and base number $n(v)$ for every vertex $v \in V(G)$.
Output: a proper 7-[3]coloring of G.

Step 1: (RBG-coloring):
Assign the base RBG-coloring to the graph G. This reduces the demand $p(v)$ by one for every vertex $v \in V(G)$. Therefore, the new demand is equal to $p_1(v) = 2$ for every vertex $v \in V(G)$. Let $G_1 = (V, E, p_1)$.

Step 2: (Suitable centers):
All suitable centers are assigned the free color, i.e. a c-free center is assigned the color c. Hence the new demands are $p_2(v) = 1$ for suitable centers and $p_2(v) = 2$ for all other vertices. The obtained graph is denoted by $G_2 = (V, E, p_2)$.

Step 3: (4-coloring of the vertices of demand 1 in G_2):
Create auxiliary graphs $G_3 = (V_3, E_3)$ and $\widetilde{G} = (V_3, \widetilde{E})$ as follows:

Let V_3 be the set of vertices of demand $p_2(v) = 1$. Note that there are no edges between vertices of V_3 in the graph G.

Define the graph G_3 by introducing edges among vertices of V_3 as follows:
Let S be a tristar in G_2 with common left center $x \notin V_3$, and the end-vertices of all three paths u, v, and z such that $u, v, z \in V_3$. If

the lengths of two paths among $(u, ..., x, ..., v)$, $(u, ..., x, ..., z)$, and $(v, ..., x, ..., z)$ are odd, then connect the end-vertices of the paths of odd length (see Figure 4).

Finally, create the graph $\widetilde{G}$ by identifying vertices of V_3 using the following rule:
Let S be a tristar in G_2 with common left center $x \notin V_3$, and the end-vertices of all three paths u, v, and z such that $u, v, z \in V_3$. If the lengths of all three paths $(u, ..., x, ..., v)$, $(u, ..., x, ..., z)$, and $(v, ..., x, ..., z)$ are even, then identify L and RD end-vertices (see Figure 4).

Color vertices of $\widetilde{G}$ with 4 colors $\{A, B, C, D\}$. Here we use the fact that $\widetilde{G}$ is a planar graph without self-loops, hence it is 4-colorable, see Lemma 5.

Use the 4-coloring of $\widetilde{G}$ to assign one color from the set $\{A, B, C, D\}$ to the vertices of G_3 in a natural way (i.e. identified vertices receive the same color).

The color assigned to vertex $v \in V(\widetilde{G})$ is denoted by $N(v)$. Clearly, this is at the same time a partial assignment to vertices of G_2. Hence the new demand is equal to $p_3(v) = p_2(v) - 1 = 0$ for every vertex $v \in V_3$ and $p_3(v) = p_2(v) = 2$ for all other vertices.

Step 4: (Extension of the obtained coloring):
Extend the assigned coloring (of Steps 1 up to 3) to the vertices of graph G, that are not completely multi-colored ($p_3(v) = 2$), in the following way. Note that the only uncolored connected subgraphs induced on vertices of demand $p_3(v) = 2$ are paths (see Lemma 4). Extend the partial coloring of G_3 using colors $\{A, B, C, D\}$ (this is possible by Lemma 6 and Lemma 7.)

3.2 Correctness proof

We show in this section that the algorithm of Section 3.1 gives a proper 7-[3]coloring of an arbitrary triangle-free hexagonal graph G. We start with an outline of the proof following the structure of the algorithm.

Step 1: RBG-coloring gives a proper 1-coloring of the graph G and reduces demands of all vertices by 1, so there is nothing to prove.

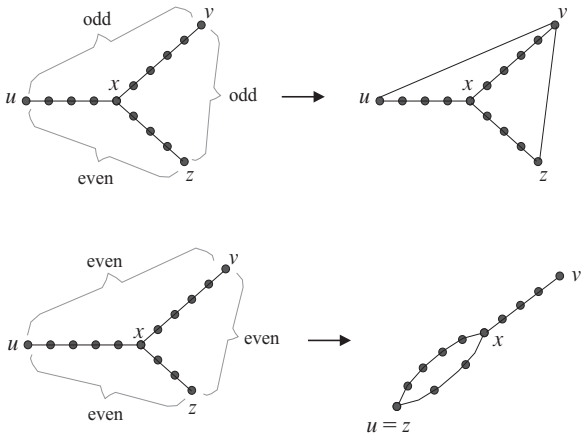

Figure 4: Identifications and new edges in Step 3 of the algorithm.

Step 2: Since a left center is assigned its c-free color only if it is not a neighbor of a right center, assigning color c to a suitable c-free center is clearly not in conflict with the previously assigned colors.

Step 3: Graph $\widetilde{G}$ is (by Lemma 5) a planar graph without self-loops, and therefore it is properly 4-[1]colorable by 4 colors $\{A, B, C, D\}$. Since only base colors $\{R, B, G\}$ were used in previous steps, no conflicts can occur.

Step 4: By Lemma 6 and Lemma 7, the partial coloring can be extended to a proper 7-[3]coloring of G.

Accordingly, every vertex $v \in V(G)$ is assigned three different colors among colors $\{R, B, G, A, B, C, D\}$ and neighbors get disjoint sets of colors, as needed.

Lemma 4 *Let G_4 be the graph induced on vertices of demand 2 in G_2. The connected components of G_4 are paths. Furthermore, the connected components of G_4 together with the neighbors in G that belong to V_3 are paths and tristars with at least one ray of length 1.*

Proof. Since every right center was assigned two colors in Steps 1 and 2, there is no right center in G_4 and consequently G_4 does not contain cycles. Therefore, the only connected components of the graph G_4 are paths and tristars (here isolated vertices are treated as paths of length 0). Let S be

a connected component that contains a left center, say v, with $p_2(v) = 2$. From the fact that v did not receive the second color in Step 2, we conclude that v is a neighbor of a right center u, and we know that $p_2(u) = 2$, so $u \notin V(S)$. Hence the degree of v in S is at most 2, and S is a path. (S can be a union of two straight paths with common vertex v.) Hence the connected components of the graph G_4 are paths.

From the above reasoning it also follows that the connected components of S together with the neighbors in G that belong to V_3 are either paths or tristars with at least one ray of length 1. ■

Lemma 5 *Graph $\widetilde{G}$ obtained in Step 3 of the algorithm is a planar graph without self-loops.*

Proof. The original graph G is an induced subgraph of triangular lattice and thus planar. It is easy to see that the identifications of vertices and the addition of new edges described in Step 3 (see Figure 4) cannot ruin the planarity of $\widetilde{G}$.

We will now see that there are no self-loops in $\widetilde{G}$. A self-loop would appear if there is a pair of adjacent vertices in a set of vertices which have to be identified. But this is not possible because all identifications involve the end-vertex of the left ray and the end-vertex of the right-down ray of a tristar. Recall that when end-vertices of a tristar are identified, they are never connected with edges in G_3. ■

Lemma 6 *Let P be a path in G_2 such that its end-vertices are in V_3 and its internal vertices have demand $p_2(v) = 2$. There exists a proper 4-[2]coloring of P that is consistent with the colors assigned to the end-vertices.*

Proof. Clear. ■

Lemma 7 *Let S be a tristar in G_2 composed of three paths: left path $(x, u_1, ..., u_k, u)$, rightup path $(x, v_1, v_2, ..., v_l, v)$, and rightdown path $(x, z_1, z_2, ..., z_m, z)$ such that $p_2(u) = p_2(v) = p_2(z) = 1$ and $p_2(s) = 2$ for all other vertices of S. There exists a proper 4-[2]coloring of $S\backslash\{u, v, z\}$ that coincides with the coloring of vertices u, v, and z in Step 3.*

Proof. We will use the fact that a tristar is bipartite. Recall that it is enough to consider tristars with at least one ray of length 1. We distinguish two cases.

First, if all three rays are of odd length, then all end-vertices are in the same set of the bipartition of the tristar. As L and RD end-vertex were

identified, the 4-coloring of Step 3 used at most 2 colors for the end-vertices. Hence the vertices of S that are in the same set of the bipartition as the end-vertices receive the two colors of the end-vertices, and the vertices of the second set of bipartition are assigned the remaining two colors.

In the second case, one or two of the other two rays are of even length. One of the end-vertices, which is at odd distance to the other two, was connected to the other two end-vertices in Step 3. Hence two end-vertices are in one, and the third end-vertex is in the other set of the bipartition of the tristar. The extension of the coloring to internal vertices of the tristar is straightforward. ■

4 Final remarks

In this article we provided an algorithm for 7-[3]coloring triangle-free hexagonal graphs. The described 7-[3]coloring can be extended to a proper $[p]$coloring with at most $\lceil (7/6)\omega(G) \rceil + o(1)$ colors of any weighted triangle-free hexagonal graph. The main idea (used in for example in [4, 5, 7, 13]) is to divide the set of colors into 7 palettes and to use the algorithm for 7-[3]coloring to define the order of color palettes from which vertices will take colors from.

Concerning the running time, it is easy to check that all steps of the algorithm run in time linear on $|V(G)|$, except for the 4-coloring of the planar graph $\widetilde{G}$, which takes a priori quadratic time [9]. It has been recently proved [1] that triangle-free planar graphs can be 3-colored in linear time. Thus, if one could prove that $\widetilde{G}$ is triangle-free (or modify the construction in such a way that the constructed graph is triangle-free), the overall running time of the algorithm would be linear.

The *odd girth* $og(G)$ of a graph G is the length of a shortest cycle of odd length in G. It is easy to see that for a triangle-free hexagonal graph G, $og(G) \geq 9$ [6]. The approximation ratio of our algorithm can be improved in terms of $og(G)$. Indeed, by assigning demand $\frac{og(G)-1}{2}$ to all the vertices belonging to a cycle of length $og(G)$, one can check that $og(G)$ colors are needed to color the vertices of that cycle. Therefore, if $p(v) \leq \frac{og(G)-1}{2}$ for all $v \in V(G)$, $\frac{og(G)}{og(G)-1}\omega(G)$ is a lower bound on the number of needed colors. For instance, if $og(G) = 9$ and $p(v) \leq 4$ for all $v \in V(G)$, the approximation ratio of our algorithm becomes $\frac{(7/6)\omega(G)}{(9/8)\omega(G)} = \frac{28}{27}$, which improves over 7/6.

Recall that both 5-[2]coloring of triangle-free hexagonal graphs [13] and a 4/3-competitive algorithm for multicoloring hexagonal graphs [12] are distributed. The 7-[3]coloring algorithm presented in Section 3 is not distributed, since it uses the existence of a 4-coloring of general planar graphs.

Nevertheless, most steps of the algorithm can be easily performed locally. Therefore, it is an interesting question whether one can find a coloring of our auxiliary graph $\widetilde{G}$ in a distributed way, using its rich structural properties.

Finally, it is a natural question to ask whether there exists an algorithm (distributed or not) for multicoloring an *arbitrary* hexagonal graph with approximation ratio 7/6.

References

[1] Z. Dvořák, K. Kawarabayashi, and R. Thomas, Three-coloring triangle-free planar graphs in linear time, Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1176-1182, 2009.

[2] W. K. Hale, Frequency Assignment: Theory and Applications, Proceedings of the IEEE **68**(12):1497-1514, 1980.

[3] F. Havet, Channel Assignment and Multicoloring of the Induced Subgraphs of the Triangular Lattice, Discrete Mathematics **233**:219-231, 2001.

[4] F. Havet and J. Žerovnik, Finding a Five Bicolouring of a Triangle-free Subgraph of the Triangular Lattice, Discrete Mathematics **244**:103-108, 2002.

[5] J. Janssen, D. Krizanc, L. Narayanan, and S. Shende, Distributed Online Frequency Assignment in Cellular Networks, Journal of Algorithms **36**:119-151, 2000.

[6] C. McDiarmid and B. Reed, Channel Assignment and Weighted Colouring, Networks **36**:114-117, 2000.

[7] L. Narayanan and S. Shende, Static Frequency Assignment in Cellular Networks, Algorithmica **29**:396-410, 2001.

[8] L. Narayanan and S. Shende, Corrigendum to Static Frequency Assignment in Cellular Networks, Algorithmica **32**:697, 2002.

[9] N. Robertson, D. P. Sanders, P. D. Seymour, and R. Thomas, The four-colour theorem, Journal of Combinatorial Theory Series B **70**:2-44, 1997.

[10] K. S. Sudeep and S. Vishwanathan, A technique for multicoloring triangle-free hexagonal graphs, Discrete Mathematics **300**:256-259, 2005.

[11] P. Šparl, S. Ubeda, and J. Žerovnik, Upper bounds for the span of frequency plans in cellular networks, International Journal of Applied Mathematics **9**(2):115-139, 2002.

[12] P. Šparl and J. Žerovnik, 2-Local 4/3-Competitive Algorithm For Multicoloring Hexagonal Graphs, Journal of Algorithms **55**:29-41, 2005.

[13] P. Šparl and J. Žerovnik, 2-local 5/4-competitive algorithm for multicoloring triangle-free hexagonal graphs, Information Processing Letters **90**:239-246, 2004.

[14] P. Šparl and J. Žerovnik, 2-local 7/6-competitive algorithm for multicoloring a sub-class of hexagonal graph, Accepted in International Journal of Computer Mathematics.

List of Figures

Index

Bibliography

Personal Bibliography

International Journals

[J1] O. Amini, F. Huc, I. Sau, and J. Žerovnik. (ℓ, k)-Routing on Plane Grids. *Journal of Interconnection Networks*, 2009. To appear.

[J2] O. Amini, S. Pérennes, and I. Sau. Hardness and Approximation of Traffic Grooming. *Theoretical Computer Science*, 2009. To appear. Online version available at `http://dx.doi.org/10.1016/j.tcs.2009.04.028`.

[J3] I. Sau and J. Žerovnik. An Optimal Permutation Routing Algorithm on Full-Duplex Hexagonal Networks. *Discrete Mathematics and Theoretical Computer Science*, 10(3):49–62, 2008.

Book Chapters

[B4] T. Cinkler, D. Coudert, M. Flammini, G. Monaco, L. Moscardelli, X. Muñoz, I. Sau, M. Shalom, and S. Zaks. *Studies in Broadband, Optical, Wireless, and Ad Hoc Networks*, chapter Traffic Grooming: Combinatorial Results and Practical Resolutions. EATCS Texts in Theoretical Computer Science. Springer, 2009. To appear.

[B5] I. Sau and J. Žerovnik. *Studies in Broadband, Optical, Wireless, and Ad Hoc Networks*, chapter Permutation Routing and (ℓ, k)-Routing on Plane Grids. EATCS Texts in Theoretical Computer Science. Springer, 2009. To appear.

International Conferences

[C6] O. Amini, D. Peleg, S. Pérennes, I. Sau, and S. Saurabh. Degree-Constrained Subgraph Problems: Hardness and Approximation Results. In *Proceedings of Workshop on Approximation and On-line Algorithms (ALGO/WAOA)*, pages 29–42, volume 5426 of LNCS, 2008.

[C7] O. Amini, S. Pérennes, and I. Sau. Hardness and Approximation of Traffic Grooming. In *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC)*, pages 561–573, volume 4835 of LNCS, 2007.

[C8] O. Amini, I. Sau, and S. Saurabh. Parameterized Complexity of the Smallest Degree-Constrained Subgraph Problem. In *Proceedings of International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 13–29, volume 5008 of LNCS, 2008.

[C9] J.-C. Bermond, D. Coudert, J. Moulierac, S. Pérennes, H. Rivano, I. Sau, and F. Solano Donado. MPLS label stacking on the line network. In *Proceedings of IFIP Networking*, volume 5550 of *LNCS*, pages 809–820, 2009.

[C10] J.-C. Bermond, D. Coudert, J. Moulierac, S. Perennes, I. Sau, and F. Solano Donado. Designing Hypergraph Layouts to GMPLS Routing Strategies. In *Proceedings of the 16th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, LNCS, 2009.

[C11] J.-C. Bermond, D. Coudert, X. Muñoz, and I. Sau. Traffic Grooming in Bidirectional WDM Ring Networks. In *Proceedings of IEEE-LEOS ICTON*, volume 3, pages 19–22, 2006.

[C12] D. Coudert, F. Giroire, and I. Sau. Edge-Simple Circuits Through 10 Ordered Vertices in Square Grids. In *Proceedings of the 20th International Workshop on Combinatorial Algorithms (IWOCA)*, 2009. To appear.

[C13] Z. Li and I. Sau. Graph Partitioning and Traffic Grooming with Bounded Degree Request Graph. In *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 2009. To appear.

[C14] G. B. Mertzios, I. Sau, and S. Zaks. A New Intersection Model and Improved Algorithms for Tolerance Graphs. In *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 2009. To appear.

[C15] X. Muñoz and I. Sau. Traffic Grooming in Unidirectional WDM Rings with Bounded Degree Request Graph. In *Proceedings of the 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 300–311, volume 5344 of LNCS, 2008.

[C16] I. Sau and D. M. Thilikos. On Self-Duality of Branchwidth in Graphs of Bounded Genus. In *Proceedings of the 8th Cologne Twente Workshop on Graphs and Combinatorial Optimization (CTW)*, pages 19–22, 2009.

[C17] I. Sau and D. M. Thilikos. Subexponential Parameterized Algorithms for Bounded-Degree Connected Subgraph Problems on Planar Graphs. In *Proceedings of DIMAP workshop on Algorithmic Graph Theory (AGT)*, volume 32 of *Electronic Notes in Discrete Mathematics*, pages 59–66, 2009.

[C18] I. Sau and J. Žerovnik. Optimal permutation routing on mesh networks. In *Proceedings of International Network Optimization Conference (INOC)*, 2007. 6 pages.

National Conferences

[N19] O. Amini, S. Pérennes, and I. Sau. Hardness of approximating the traffic grooming problem. In *Proceedings of 9ème Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel)*, pages 45–48, 2007.

Submitted for Publication

[S20] O. Amini, D. Peleg, S. Pérennes, I. Sau, and S. Saurabh. Degree-Constrained Subgraph Problems: Hardness and Approximation. Manuscript submitted to journal, 2009.

[S21] J.-C. Bermond, C. J. Colbourn, L. Gionfriddo, G. Quattrocchi, and I. Sau. Drop Cost and Wavelength Optimal Two-Period Grooming with Ratio 4. Manuscript submitted to journal, 2008.

[S22] G. B. Mertzios, I. Sau, and S. Zaks. A New Intersection Model and Improved Algorithms for Tolerance Graphs. Manuscript submitted to journal, 2009.

[S23] G. B. Mertzios, I. Sau, and S. Zaks. The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete. Manuscript submitted to conference, 2009.

[S24] J. Rué, I. Sau, and D. M. Thilikos. Dynamic Programming for Graphs on Surfaces. Manuscript submitted to conference, 2009.

[S25] I. Sau and D. M. Thilikos. Subexponential Parameterized Algorithms for Degree-Constrained Subgraph Problems on Planar Graphs. Manuscript submitted to journal, 2009.

[S26] I. Sau, P. Šparl, and J. Žerovnik. 7-[3]coloring Algorithm for Triangle-free Hexagonal Graphs. Manuscript submitted to journal, 2009.

In Preparation

[P27] O. Amini, I. Sau, and S. Saurabh. Parameterized Complexity of Finding Small Degree-Constrained Subgraphs. Journal article in preparation, 2009.

[P28] J.-C. Bermond, X. Muñoz, and I. Sau. Traffic Grooming in Bidirectional WDM Ring Networks. Journal article in preparation, 2009.

[P29] X. Muñoz, Z. Li, and I. Sau. Traffic Grooming in Unidirectional WDM Rings with Bounded-degree Request Graph. Journal article in preparation, 2009.

General Bibliography

[30] L. Addario-Berry, K. Dalal, and B. Reed. Degree constrained subgraphs. *Discrete Applied Mathematics*, 156(7):1168–1174, 2008.

[31] N. Alon, S. Friedland, and G. Kalai. Every 4-regular graph plus an edge contains a 3-regular subgraph. *Journal of Combinatorial Theory Series B*, 37:92–93, 1984.

[32] N. Alon, S. Friedland, and G. Kalai. Regular subgraphs of almost regular graphs. *Journal of Combinatorial Theory Series B*, 37:79–91, 1984.

[33] N. Alon, V. Teague, and N. C. Wormald. Linear Arboricity and Linear k-Arboricity of Regular Graphs. *Graphs and Combinatorics*, 17(1):11–16, 2001.

[34] N. Alon, R. Yuster, and U. Zwick. Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs. In *Proceedings of the 26th annual ACM symposium on Theory of Computing (STOC)*, pages 326–335, 1994.

[35] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 7–15, 2001.

[36] R. Andersen and K. Chellapilla. Finding Dense Subgraphs with Size Bounds. In *Proceedings of the 6th International Workshop on Algorithms and Models for the Web-Graph (WAW)*, pages 25–37, volume 5427 of LNCS, 2009.

[37] R. P. Anstee. Minimum vertex weighted deficiency of (g, f)-factors: a greedy algorithm. *Discrete Applied Mathemathics*, 44(1-3):247–260, 1993.

[38] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – a survey. *BIT*, 25(1):2–23, 1985.

[39] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25:272–289, 1996.

[40] B. Beauquier, J.-C. Bermond, L. Gargano, P. Hell, S. Perennes, and U. Vaccaro. Graph problems arising from Wavelength–Routing in All–Optical Networks. In *Proceedings of Workshop on Optics and Computer Science (WOCS)*, 1997.

[41] E. A. Bender, Z. Gao, and L. B. Richmond. The map asymptotics constant t_g. *Electronic Journal of Combinatorics*, 15(1):R51, 8 psges, 2008.

[42] C. Berge. *Graphs and Hypergraphs*. North-Holland, 1973.

[43] J.-C. Bermond, L. Braud, and D. Coudert. Traffic Grooming on the Path. *Theoretical Computer Science*, 384(2-3):139–151, 2007.

[44] J.-C. Bermond and S. Ceroi. Minimizing SONET ADMs in unidirectional WDM rings with grooming ratio 3. *Networks*, 41(2):83–86, 2003.

[45] J.-C. Bermond, C. J. Colbourn, D. Coudert, G. Ge, A. C. H. Ling, and X. Muñoz. Traffic grooming in unidirectional WDM rings with grooming ratio $C = 6$. *SIAM Journal on Discrete Mathematics*, 19(2):523–542, 2005.

[46] J.-C. Bermond, C. J. Colbourn, A. C. H. Ling, and M. L. Yu. Grooming in unidirectional rings: $K_4 - e$ designs. *Discrete Mathematics*, 284:67–72, 2004.

[47] J.-C. Bermond and D. Coudert. Traffic Grooming in Unidirectional WDM Ring Networks using Design Theory. In *Proceedings of IEEE International Conference on Communications (ICC)*, volume 2, pages 1402–1406, 2003.

[48] J.-C. Bermond and D. Coudert. *The CRC Handbook of Combinatorial Designs (2nd edition)*, volume 42 of *Discrete Mathematics and Its Applications*, chapter VI.27, Grooming, pages 493–496. CRC Press, C.J. Colbourn and J.H. Dinitz edition, 2006.

[49] J.-C. Bermond, D. Coudert, and B. Lévêque. Approximations for All-to-all Uniform Traffic Grooming on Unidirectional Rings. *Journal of Interconnection Networks*, 9(4):471–486, 2008.

[50] J.-C. Bermond, D. Coudert, and X. Muñoz. Traffic Grooming in Unidirectional WDM Ring Networks: the all-to-all unitary case. In *Proceedings of the 7th IFIP Working Conference on Optical Network Design and Modelling*, pages 1135–1153, 2003.

[51] J.-C. Bermond, D. Coudert, and M.-L. Yu. On DRC-Covering of K_n by cycles. *Journal of Combinatorial Designs*, 11(2):100–112, 2003.

[52] J.-C. Bermond, J.-L. Fouquet, M. Habib, and B. Péroche. On linear k-arboricity. *Discrete Mathematics*, 52(2-3):123–132, 1984.

[53] J.-C. Bermond and C. Peyrat. Induced Subgraphs of the Power of a Cycle. *SIAM Journal on Discrete Mathematics*, 2(4):452–455, 1989.

[54] O. Bernardi and J. Rué. Counting simplicial decompositions in surfaces with boundaries. Manuscript available at `http://lanl.arxiv.org/abs/0901.1608`.

[55] R. Berry and E. Modiano. Reducing electronic multiplexing costs in SONET/WDM rings with dynamically changing traffic. *IEEE Journal on Selected Areas in Communications*, 18:1961–1971, 2000.

[56] A. Björklund and T. Husfeldt. Finding a Path of Superlogarithmic Length. *SIAM Journal on Computing*, 32(6):1395–1402, 2003.

[57] H. L. Bodlaender. Dynamic Programming on Graphs with Bounded Treewidth. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 105–118, volume 317 of LNCS, 1988.

[58] H. L. Bodlaender. A Tourist Guide through Treewidth. *Acta Cybernetica*, 11(1-2):1–22, 1993.

[59] B. Bollobás and G. Brightwell. Long Cycles in Graphs with no Subgraphs of Minimal Degree 3. *Discrete Mathematics*, 75:47–53, 1989.

[60] V. Bonifaci, U. D. Iorio, and L. Laura. The complexity of uniform Nash equilibria and related regular subgraph problems. *Theoretical Computer Science*, 401(1-3):144–152, 2008.

[61] D. Bryant, P. Adams, and M. Buchanan. A survey on the existence of G-designs. *Journal of Combinatorial Designs*, 16:373–410, 2008.

[62] S. Cabello and B. Mohar. Finding shortest non-separating and non-contractible cycles for topologically embedded graphs. *Discrete and Computational Geometry*, 37(2):213–235, 2007.

[63] A. Caprara and R. Rizzi. Packing triangles in bounded degree graphs. *Information Processing Letters*, 84(4):175–180, 2002.

[64] D. M. Cardoso, M. Kamiński, and V. Lozin. Maximum k-regular induced subgraphs. *Journal of Combinatorial Optimization*, 14(4):455–463, 2007.

[65] Y. Caro and J. Schönheim. Decompositions of trees into isomorphic subtrees. *Ars Combininatorica*, 9:119–130, 1980.

[66] P. A. Catlin. Supereulerian graphs: a survey. *Journal of Graph Theory*, 16(2):177–196, 1992.

[67] L. S. Chandran. A high girth graph construction. *SIAM Journal on Discrete Mathematics*, 16(3):366–370, 2003.

[68] F. Cheah and D. G. Corneil. The Complexity of Regular Subgraph Recognition. *Discrete Applied Mathematics*, 27:59–68, 1990.

[69] A. L. Chiu and E. H. Modiano. Traffic grooming algorithms for reducing electronic multiplexing costs in WDM ring networks. *IEEE/OSA Journal of Lightwave Technology*, 18(1):2–12, 2000.

[70] B. Chor, M. Fellows, M. A. Ragan, I. Razgon, F. Rosamond, and S. Snir. Connected Coloring Completion for General Graphs: Algorithms and Complexity. In *Proceedings of the 13th Annual International Computing and Combinatorics Conference (COCOON)*, pages 75–85, volume 4598 of LNCS, 2007.

[71] T. Chow and P. Lin. Private communication.

[72] T. Chow and P. Lin. The ring grooming problem. *Networks*, 44(3):194–202, 2004.

[73] V. Chvátal, H. Fleischner, J. Sheehan, and C. Thomassen. Three-regular Subgraphs of Four Regular Graphs. *Journal of Graph Theory*, 3:371–386, 1979.

[74] T. Cinkler. Traffic- and λ-grooming. *IEEE Network*, 17(2):16–21, 2003.

[75] C. Colbourn and J. Dinitz, editors. *Handbook of Combinatorial Designs*. Number 0. Chapman & Hall/CRC, 2nd edition, 2006.

[76] C. J. Colbourn, H.-L. Fu, G. Ge, A. C. H. Ling, and H.-C. Lu. Minimizing SONET ADMs in Unidirectional WDM Rings with Grooming Ratio Seven. *SIAM Journal on Discrete Mathematics*, 23(1):109–122, 2008.

[77] C. J. Colbourn, A. C. H. Ling, and G. Quattrocchi. Minimum embedding of P_3-designs into $(K_4 - e)$-designs. *Journal of Combinatorial Designs*, 11:352–366, 2003.

[78] C. J. Colbourn, G. Quattrocchi, and V. R. Syrotiuk. Grooming for two-period optical networks. *Networks*, 52(4):307–324, 2008.

[79] C. J. Colbourn, G. Quattrocchi, and V. R. Syrotiuk. Lower bounds for two-period grooming via linear programming duality. *Networks*, 52(4):299–306, 2008.

[80] C. J. Colbourn and A. Rosa. Quadratic leaves of maximal partial triple systems. *Graphs and Combinatorics*, 2:317–337, 1986.

[81] C. J. Colbourn and A. Rosa. *Triple systems*. Oxford University Press, Oxford and New York, 1999.

[82] C. J. Colbourn and P.-J. Wan. Minimizing Drop Cost for SONET/WDM Networks with 1/8 Wavelength Requirements. *Networks*, 37(2):107–116, 2001.

[83] W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. John Wiley and Sons, New York, 1998.

[84] B. Courcelle. The Monadic Second-Order Logic of Graphs: Definable Sets of Finite Graphs. In *Proceedings of the 14th International Workshop on Graph-theoretic Concepts in Computer Science (WG)*, volume 344 of *LNCS*, pages 30–53, 1988.

[85] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Accessible at `http://www.nada.kth.se/~viggo/wwwcompendium`.

[86] G. Călinescu, O. Frieder, and P.-J. Wan. Minimizing electronic line terminals for automatic ring protection in general WDM optical networks. *IEEE Journal of Selected Areas on Communications*, 20(1):183–189, 2002.

[87] A. Dawar, M. Grohe, and S. Kreutzer. Locally Excluding a Minor. In *Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 270–279, 2007.

[88] A. Dawar, M. Grohe, S. Kreutzer, and N. Schweikardt. Approximation schemes for first-order definable optimisation problems. In *Proceedings of the 21st IEEE Symposium on Logic in Computer Science (LICS)*, pages 411–420, 2006.

[89] D. de Werra. Equitable colorations of graphs. *RAIRO R-3*, pages 3–8, 1971.

[90] E. Demaine, M. Hajiaghayi, and K. C. Kawarabayashi. Algorithmic Graph Minor Theory: Decomposition, Approximation and Coloring. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 637–646, 2005.

[91] E. Demaine and M. T. Hajiaghayi. Equivalence of Local Treewidth and Linear Local Treewidth and its Algorithmic Applications. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 840–849, 2004.

[92] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs. *Journal of the ACM*, 52(6):866–893, 2005.

[93] E. D. Demaine and M. Hajiaghayi. Bidimensionality: New Connections between FPT Algorithms and PTASs. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 590–601, 2005.

[94] E. D. Demaine, M. Hajiaghayi, and D. M. Thilikos. The Bidimensional Theory of Bounded-Genus Graphs. *SIAM Journal on Discrete Mathematics*, 20(2):357–371, 2006.

[95] R. Diestel. *Graph Theory*. Springer-Verlag, 2005.

[96] F. Dorn. *Designing Subexponential Algorithms: Problems, Techniques and Structures*. PhD thesis, University of Bergen, 2007.

[97] F. Dorn, F. V. Fomin, and D. M. Thilikos. Fast Subexponential Algorithm for Non-local Problems on Graphs of Bounded Genus. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 4059 of *LNCS*, pages 172–183, 2006.

[98] F. Dorn, F. V. Fomin, and D. M. Thilikos. Subexponential parameterized algorithms. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 15–27, volume 4596 of LNCS, 2007.

[99] F. Dorn, F. V. Fomin, and D. M. Thilikos. Catalan structures and dynamic programming in H-minor-free graphs. In *Proceedings of the 19th annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 631–640, 2008.

[100] F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient Exact Algorithms on Planar Graphs: Exploiting Sphere Cut Branch Decompositions. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA)*, pages 95–106, volume 3669 of LNCS, 2005.

[101] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, New York, 1999.

[102] R. Dutta, A. E. Kamal, and G. N. Rouskas, editors. *Traffic Grooming for Optical Networks: Foundations, Techniques and Frontiers*. Optical Networks. Springer, 2008.

[103] R. Dutta and N. Rouskas. A survey of virtual topology design algorithms for wavelength routed optical networks. *Optical Networks*, 1(1):73–89, 2000.

[104] R. Dutta and N. Rouskas. Traffic Grooming in WDM Networks: Past and Future. *IEEE Network*, 16(6):46–56, 2002.

[105] T. Eilam, S. Moran, and S. Zaks. Lightpath arrangement in survivable rings to minimize the switching cost. *IEEE Journal of Selected Areas on Communications*, 20(1):172–182, 2002.

[106] D. Eppstein. Diameter and Tree-width in Minor-closed Graph Families. *Algorithmica*, 27(3-4):275–291, 200.

[107] D. Eppstein. Dynamic Generators of Topologically Embedded Graphs. In *Proceedings of the 14th annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 599–608, 2003.

[108] L. Epstein and A. Levin. Better Bounds for Minimizing SONET ADMs. In *Proceedings of the Workshop on Approximation and On-line Algorithms (WAOA)*, pages 281–294, volume 3351 of LNCS, 2004.

[109] P. Erdős, R. J. Faudree, A. Gyárfás, and R. H. Schelp. Cycles in Graphs Without Proper Subgraphs of Minimum Degree 3. *Ars Combinatorica*, 25(B):195–201, 1988.

[110] P. Erdős, R. J. Faudree, C. C. Rousseau, and R. H. Schelp. Subgraphs of Minimal Degree k. *Discrete Mathematics*, 85(1):53–58, 1990.

[111] P. Erdös and H. Sachs. Reguläre graphe gegebener taillenweite mit minimaler knotenzahl. *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe*, 12:251–257, 1963.

[112] U. Feige, G. Kortsarz, and D. Peleg. The Dense k-Subgraph Problem. *Algorithmica*, 29(3):410–421, 2001.

[113] M. Fellows, D. Hermelin, F. Rosamond, and S. Vialette. On the Parameterized Complexity of Multiple-Interval Graph Problems. *Theoretical Computer Science*, 410(1):53–61, 2009.

[114] P. Flajolet and M. Noy. Analytic combinatorics of non-crossing configurations. *Discrete Mathematics*, 204(1-3):203–229, 1999.

[115] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2008.

[116] M. Flammini, G. Monaco, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the traffic grooming problem in tree and star networks. *Journal of Parallel and Distributed Computing*, 68(7):939–948, 2008.

[117] M. Flammini, G. Monaco, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the Traffic Grooming Problem with Respect to ADMs and OADMs. In *Proceedings of the 14th International Conference on Parallel and Distributed Computing (Euro-Par)*, pages 920–929, 2008.

[118] M. Flammini, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the traffic grooming problem. *Journal of Discrete Algorithms*, 6(3):472–479, 2008.

[119] M. Flammini, M. Shalom, and S. Zaks. On minimizing the number of adms in a general topology optical network. In *Proceedings of the 20th International Workshop on Distributed Algorithms (DISC)*, 2006.

[120] M. Flammini, M. Shalom, and S. Zaks. On minimizing the number of adms - tight bounds for an algorithm without preprocessing. *Journal of Parallel and Distributed Computing*, 67(4):448–455, 2007.

[121] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer Verlag, 2006.

[122] F. V. Fomin and D. M. Thilikos. Fast Parameterized Algorithms for Graphs on Surfaces: Linear Kernel and Exponential Speed-Up . In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3142 of *LNCS*, pages 581–592, 2004.

[123] F. V. Fomin and D. M. Thilikos. New upper bounds on the decomposability of planar graphs. *Journal of Graph Theory*, 51(1):53–81, 2005.

[124] F. V. Fomin and D. M. Thilikos. Dominating Sets in Planar Graphs: Branch-Width and Exponential Speed-Up. *SIAM Journal on Computing*, 36(2):281–309, 2006.

[125] F. V. Fomin and D. M. Thilikos. On Self Duality of Pathwidth in Polyhedral Graph Embeddings. *Journal of Graph Theory*, 55(42-54), 2007.

[126] M. Frick and M. Grohe. Deciding First-Order Properties of Locally Tree-Decomposable Structures. *Journal of the ACM*, 48(6):1184–1206, 2001.

[127] H. L. Fu and C. A. Rodger. Forest leaves and 4-cycles. *Journal of Graph Theory*, 33:161–166, 2000.

[128] M. Fürer and B. Raghavachari. Approximating the minimum-degree spanning tree to within one from the optimal degree. In *Proceedings of the 3rd annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 317–324, 1992.

[129] M. Fürer and B. Raghavachari. Approximating the minimum-degree steiner tree to within one of optimal. *Journal of Algorithms*, 17(3):409–423, 1994.

[130] H. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the 15th annual ACM symposium on Theory of Computing (STOC)*, pages 448–456, 1983.

[131] Z. Gao. The number of rooted triangular maps on a surface. *Journal of Combinatorial Theory, Series B*, 52(2):236–249, 1991.

[132] M. Garey and D. Johnson. *Computers and Intractability.* W.H. Freeman, 1979.

[133] O. Gerstel, P. Lin, and G. Sasaki. Wavelength assignment in a WDM ring to minimize cost of embedded SONET rings. In *IEEE INFOCOM*, pages 94–101, 1998.

[134] O. Gerstel, R. Ramaswami, and G. Sasaki. Cost Effective Traffic Grooming in WDM Rings. In *Proceedings of IEEE INFOCOM*, pages 69–77, 1998.

[135] M. X. Goemans. Minimum Bounded-Degree Spanning Trees. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 273–282, 2006.

[136] O. Goldschmidt, D. Hochbaum, A. Levin, and E. Olinick. The SONET edge-partition problem. *Networks*, 41(1):13–23, 2003.

[137] M. Grohe. Local Tree-width, Excluded Minors and Approximation Algorithms. *Combinatorica*, 23(4):613–632, 2003.

[138] M. Grohe. Logic, graphs, and algorithms. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(091), 2007.

[139] Q.-P. Gu and H. Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. *ACM Transactions on Algorithms*, 4(3), 2008.

[140] M. Hajiaghayi. *The bidimensionality theory and its algorithmic applications.* PhD thesis, Massachusetts Institute of Technology, Cambridge, USA, 2005.

[141] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

[142] M. R. Henzinger, S. Rao, and H. N. Gabow. Computing Vertex Connectivity: New Bounds from Old Techniques. *Journal of Algorithms*, 34(2):222–250, 2000.

[143] I. Holyer. The NP-Completeness of Some Edge-Partition Problems. *SIAM Journal on Computing*, 10(4):713–717, 1981.

[144] Y. Hong-Hsu, S. Lee, and B. Mukherjee. Traffic grooming and delay constrained multicast routing in IP over WDM networks. In *Proceedings of IEEE International Conference on Communications (ICC)*, pages 5246–5251, 2008.

[145] J. Hu. Traffic Grooming in WDM Ring Networks: A Linear Programming Solution. *OSA Journal of Optical Networks*, 1(11):397–408, 2002.

[146] S. Huang, R. Dutta, and G. Rouskas. Traffic Grooming in Path, Star, and Tree Networks: Complexity, Bounds, and Algorithms. *IEEE Journal on Selected Areas in Communications*, 24(4):66–82, 2006.

[147] C. A. J. Hurkens and A. Schrijver. On the size of systems of sets every t of which have an sdr, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal on Discrete Mathematics*, 2(1):68–72, 1989.

[148] R. Impagliazzo, R. Paturi, and F. Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. Special issue of FOCS 1998.

[149] V. Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Information Processing Letters*, 37:27–35, 1991.

[150] D. Karger, R. Motwani, and G. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

[151] A. Kézdy. *Studies in Connectivity.* PhD thesis, University of Illinois at Urbana-Champaign, 1991.

[152] S. Khot. Ruling Out PTAS for Graph Min-Bisection, Dense k-Subgraph, and Bipartite Clique. *SIAM Journal on Computing*, 36(4):1025–1071, 2006.

[153] P. N. Klein, R. Krishnan, B. Raghavachari, and R. Ravi. Approximation Algorithms for Finding Low-Degree Subgraphs. *Networks*, 44(3):203–215, 2004.

[154] M. Köhn. A New Efficient Online-Optimization Approach for SDH/SONET-WDM Multi Layer Networks. In *Proceedings of Optical Fiber Communication Conference (OFC)*, 2006.

[155] G. Kreweras. Sur les partitions non croisées d'un cercle. *Discrete Mathematics*, 1:333–350, 1972.

[156] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Oxford University Press, 1976.

[157] L. Liu, X. Li, P.-J. Wan, and O. Frieder. Wavelength Assignment in WDM Rings to Minimize SONET ADMs. In *Proceedings of IEEE INFOCOM*, pages 1020–1025, 2000.

[158] L. Lovász and M. Plummer. *Matching Theory.* Annals of Discrete Mathematics 29, North-Holland, 1986.

[159] C. Lund and M. Yannakakis. The Approximation of Maximum Subgraph Problems. *Proceedings of the 20th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 40–51, volume 700 of LNCS, 1993.

[160] L. Mathieson, E. Prieto, and P. Shaw. Packing edge disjoint triangles: A parameterized view. In *Proceedings of the International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 127–137, volume 3162 of LNCS, 2004.

[161] L. Mathieson and S. Szeider. The Parameterized Complexity of Regular Subgraph Problems and Generalizations. In *Proceedings of the 14th Computing: The Australasian Theory Symposium (CATS)*, pages 79–86, volume 77 of CRPIT, 2008.

[162] E. Modiano and P. Lin. Traffic grooming in WDM networks. *IEEE Communications Magazine*, 39(7):124–129, 2001.

[163] B. Mohar and C. Thomassen. *Graphs on surfaces*. John Hopkins University Press, 2001.

[164] H. Moser and D. M. Thilikos. Parameterized Complexity of Finding Regular Induced Subgraphs. *Journal of Discrete Algorithms*, 7(2):181–190, 2009.

[165] J. Munro and V. Raman. Succinct Representation of Balanced Parentheses and Static Trees. *SIAM Journal on Computing*, 31(3):762–776, 2001.

[166] R. Niedermeier. *Invitation to Fixed Parameter Algorithms*. Oxford University Press, 2006.

[167] C. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[168] J. P. Petersen. Die Theorie der Regulären Graphs. *Acta Mathematica*, 15:193–220, 1891.

[169] M. Plantholt. The chromatic index of graphs with a spanning star. *Journal of Graph Theory*, 5:45–53, 1981.

[170] G. Quattrocchi. Embedding path designs in 4-cycle systems. *Discrete Mathematics*, 255:349–356, 2002.

[171] R. Ravi, M. Marathe, S. Ravi, D. Rosenkrantz, and H. Hunt III. Approximation Algorithms for Degree-Constrained Minimum-Cost Network-Design Problems. *Algorithmica*, 31(1):58–78, 2001.

[172] O. Richard. The Number of Trees. *Annals of Mathematics*, Second Series 49(3):583–599, 1948.

[173] N. Robertson and P. Seymour. Graph Minors X. Obstructions to Tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

[174] N. Robertson and P. Seymour. Graph Minors XII. Distance on a Surface. *Journal of Combinatorial Theory, Series B*, 64:240–272, 1995.

[175] N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, 1994.

[176] N. Robertson and P. D. Seymour. Graph Minors XIII. The Disjoint Paths Problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

[177] N. Robertson and P. D. Seymour. Graph Minors XVI. Excluding a Non-Planar Graph. *Journal of Combinatorial Theory, Series B*, 89(1):43–76, 2003.

[178] N. Robertson and P. D. Seymour. Graph Minors XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

[179] A. Rosa and W. D. Wallis. Premature sets of 1-factors, or, how not to schedule round-robin tournaments. *Discrete Applied Mathematics*, 4:291–297, 1982.

[180] P. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[181] M. Shalom, W. Unger, and S. Zaks. On the Complexity of the Traffic Grooming Problem in Optical Networks. In *Proceedings of the 4th International Conference on Fun with Algorithms*, pages 262–271, volume 4475 of LNCS, 2007.

[182] M. Shalom and S. Zaks. A $10/7 + \epsilon$ approximation scheme for minimizing the number of ADMs in SONET rings. In *Proceedings of BROADNETS*, pages 254–262, 2004.

[183] Y. Shiloach. Another look at the degree constrained subgraph problem. *Information Processing Letters*, 12(2):89–92, 1981.

[184] A. Somani. *Survivability & Traffic Grooming in WDM Optical Networks*. Cambridge Press, 2006.

[185] I. A. Stewart. Finding Regular Subgraphs in Both Arbitrary and Planar Graphs. *Discrete Applied Mathematics*, 68(3):223–235, 1996.

[186] D. H. Su and D. W. Griffith. Standards activities for MPLS over WDM networks. *Optical Networks Magazine*, 1(3), 2000.

[187] A. Suzuki and T. Tokuyama. Dense subgraph problem revisited. In *Proceedings of the Joint Workshop "New Horizons in Computing" and "Statistical Mechanical Approach to Probabilistic Information Processing"*, 2005.

[188] J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10(4):529–550, 1997.

[189] C. Thomassen. The graph genus problem is NP-complete. *Journal of Algorithms*, 10(4):568–576, 1989.

[190] C. Thomassen. Two-coloring the edges of a cubic graph such that each monochromatic component is a path of length at most 5. *Journal of Combinatorial Theory, Series B*, 75(1):100–109, 1999.

[191] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2003.

[192] P.-J. Wan, G. Calinescu, L. Liu, and O. Frieder. Grooming of arbitrary traffic in SONET/WDM BLSRs. *IEEE Journal of Selected Areas in Communications*, 18(10):1995–2003, 2000.

[193] J. Wang, W. Cho, V. R. Vemuri, and B. Mukherjee. Improved approaches for cost-effective traffic grooming in WDM ring networks: ILP formulations and single-hop and multihop connections. *IEEE/OSA Journal of Lightwave Technology*, 19(11):1645–1653, 2001.

[194] R. M. Wilson. Decomposition of complete graphs into subgraphs isomorphic to a given graph. *Congressus numerantium*, 15:647–659, 1976.

[195] S. Win. On a Connection Between the Existence of k-Trees and the Toughness of a Graph. *Graphs and Combinatorics*, 5(1):201–205, 1989.

[196] E. B. Yavorskii. Representations of directed graphs and ψ-transformations. *Theoretical and Applied Questions of Differential Equations and Algebra*, pages 247–250, 1978. A. N. Sharkovskii (Editor).

[197] X. Zhang and C. Qiao. An effective and comprehensive approach for traffic grooming and wavelength assignment in SONET/WDM rings. *IEEE/ACM Transactions on Networking*, 8(5):608–617, 2000.

[198] K. Zhu and B. Mukherjee. A review of traffic grooming in WDM optical networks: Architectures and challenges. *Optical Networks Magazine*, 4(2):55–64, 2003.

[199] K. Zhu, H. Zhu, and B. Mukherjee. Traffic engineering in multigranularity heterogeneous optical WDM mesh networks through dynamic traffic grooming. *IEEE Network*, 17(2):8–15, 2003.

[200] K. Zhu, H. Zhu, and B. Mukherjee. *Traffic Grooming in Optical WDM Mesh Networks*. Springer, 2005.