WORD-PROCESSING-BASED ROUTING FOR CAYLEY GRAPHS

Daniela Aguirre Guerrero

Doctoral Thesis

WORD-PROCESSING-BASED ROUTING FOR CAYLEY GRAPHS

DANIELA AGUIRRE GUERRERO

2019

Doctoral Program in Technology

Supervised by:
Dr. Pere Vilà Talleda and Dr. Lluís Fàbrega Soler

Thesis submitted to the University of Girona in fulfillment of the requirements for the degree of Doctor of Philosophy

CERTIFICAT DE DIRECCIÓ DE TESI

Dr. Pere Vilà Talleda, Dr. Lluís Fàbrega Soler, del Departament d'Arquitectura i Tecnologia de Computadors de la Universitat de Girona,

DECLAREM:

Que el treball titulat *Word-Processing-based Routing for Cayley Graphs*, que presenta Daniela Aguirre Guerrero per a l'obtenció del títol de doctor, ha estat realitzat sota la nostra direcció i que compleix els requisits per poder optar a Menció Internacional.

I, perquè així consti i tingui els efectes oportuns, signem aquest document.

Girona, Gener 2019

Dr. Pere Vilà Talleda

Dr. Lluís Fàbrega Soler

ACKNOWLEDGMENTS

The work presented throughout this manuscript would not have been possible without the support of some people and institutions. I would like to express my gratitude to all of them.

Firstly, I would like to acknowledge the financial support of the Mexican Council for Science and Technology (CONACyT) for the award of the PhD grant *CONACyT-SENER 2015-409697*. Also, I thank the Broadband Communications and Distributed Systems (BCDS) group of the University of Girona (UdG) for the financial support to attending different scientific events and research stays.

I would like to express my gratitude to my advisors, Dr. Pere Vilà and Dr. Lluís Fàbrega, for their continuous support, patience and motivation. Their guidance helped me in all the time. I am deeply grateful to Professor. Jose Luis Marzo, who gave me the opportunity to work in the BCDS group.

One of the most important experiences during my PhD was my research stay at the Combinatorics, Optimization, and Algorithms for Telecommunications (COATI) team of the French Institute for Research in Computer Science and Automation (INRIA). I would like to thank Dr. David Coudert and Dr. Guillaume Ducoffe, who host me at INRIA and provided rigorous and valuable feedback not only during my research stay but also in the last months. Also, I thank the valuable feedback that the anonymous reviewers of this manuscript (and the papers that resulted from this research work) have provided throughout these years.

También me gustaría agradecer a mis amigos de México: Josué Martínez, Salvador Peña, Fernando Guerrero e Ismael Robles, quienes en distintas maneras me acompañaron y apoyaron durante este proceso. Por supuesto, también tengo mucho que agradecer a las personas con quienes compartí la vida en Girona. A Ana Mar Oropeza, Silvia Oviedo, Silvia Baldiris y Oscar Azorin, gracias por su incondicional amistad y compañía, la cual estoy segura continuará sin importar el lugar en donde nos encontremos.

Finalmente, quisiera expresar mi más profunda gratitud a mis padres Juan Manuel y María Elena, y a mis hermanos Liz, Juan y Julio por su incondicional amor y apoyo.

PUBLICATIONS

The work developed in this Thesis led to the following publications:

JOURNAL ARTICLES

[DAM'18] **D. Aguirre-Guerrero**, G. Ducoffe, Ll. Fàbrega, P. Vilà and D. Coudert, "Low Time Complexity Algorithms for Path Computation in Cayley Graphs," in *Discrete Applied Mathematics*, 2018, ISSN: 0166-218X, DOI: 10.1016/j.dam.2018.12.005.

[TNET'18] **D. Aguirre-Guerrero**, M. Camelo, Ll. Fàbrega and P. Vilà, "WMGR: A Generic and Compact Routing Scheme for Data Center Networks," in *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 356-369. Feb. 2018. ISSN: 0166-218X, DOI: 10.1109/TNET.2017.2779866.

PEER-REVIEWED CONFERENCES AND WORKSHOPS

[CITS'18] **D. Aguirre-Guerrero**, A. Mañosa, Ll. Fàbrega and P. Vilà, "Evaluation of Cayley Graphs for Parallel Computer Systems," *2018 International Conference on Computer, Information and Telecommunication Systems (CITS 2018)*, Colmar, France, 2018. ISBN: 978-1-5386-4599-4, DOI: 10.1109/CITS.2018.8440157.

[WEN'16] **D. Aguirre-Guerrero**, Ll. Fàbrega, P. Vilà and M. Camelo, "Compact Greedy Routing in Large-scale Networks using Word-metric Spaces," *1st. International Workshop on Elastic Networks Design and Optimization*, Cartagena, Spain, 2016.

OTHER CONFERENCES AND WORKSHOPS

[SBC'17] **D. Aguirre-Guerrero**, Ll. Fàbrega, and P. Vilà, "Encaminamiento de Información en Redes de Comunicación de Gran Escala," *6to. Simposio de Becarios CONACyT en Europa*, Strasbourg, Fran-ce 2017.

[CPRG'16] **D. Aguirre-Guerrero**, Ll. Fàbrega and P. Vilà, "Greedy Geometric Routing in Word-metric Spaces," *1st. Conference of Predoctorals Researches*, Girona, Spain, 2016. ISBN: 978-8-48458-502-2.

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ACRONYMS

AGT	Automatic Group Theory
BCDS	Broadband Communications and Distributed Systems
BFS	Breadth-First Search
CG	Cayley Graph
COATI	Combinatorics, Optimization, and Algorithms for Telecommunications
CONACyT	Mexican Council for Science and Technology
DCN	Data Center Network
FSA	Finite State Automata
GCR	Generalized Chordal Ring
GRWMS	Geometric Routing with Word-Metric Spaces
HPC	High Performance Computing
INRIA	French Institute for Research in Computer Science and Automation
KBMAG	Knuth-Bendix on Monoids and Automatic Groups
MA	Multiplier Automata
MWP	Minimum Word Problem
PCAACG	Path Computation Algorithm for Abelian Cayley Graphs
PIN	Processor Interconnection Network
RCRR	Routing based on Chordal Ring Representation
RPS	Routing based on Permutation Sort
SAG	*ShortLex* Automatic Group
SAS	*ShortLex* Automatic Structure
SFA	Sims Factoring Algorithm
UdG	University of Girona
UGAL	Universal Globally-Adaptive Load-balanced Routing
WA	Word-Acceptor
WD	Word-Difference
WDA	Word-Difference Automaton

WPR	Word-Processing-based Routing
WSN	Wireless Sensor Network
2D	2-Dimensional

CONTENTS

ABSTRACT

Cayley Graphs (CGs) are a geometric representation of algebraic groups that have been used as topologies of a wide variety of communication networks, e.g. Processor Interconnection Networks (PINs), Wireless Sensor Networks (WSNs), Data Center Networks (DCNs), etc. The reason is that their properties of node-transitivity, link-connectivity and low average distance between nodes enable high performance and robustness in large-scale networks.

Most of communication networks, whose topologies are defined by CGs, apply traditional routing schemes, such as the *Valiant Routing* and the *Universal Globally-Adaptive Load-balanced Routing (UGAL)*. These routing schemes employ topology-agnostic algorithms for path computation, such as the Bellman-Ford and Dijsktra algorithms. These algorithms receive the whole graph as input, which results in high space and time complexity for the routing schemes using them.

In contrast, routing schemes and path computation algorithms dedicated to CGs have been designed to achieve low space and time complexity taking advantage of the aforementioned properties of CGs. These proposals can be classified into: 1) generic, i.e. that works on several families of CGs; and 2) specialized on a family of CGs. The challenge of achieving low time and space complexity is major for generic proposals as they must work on CGs with different topological structures.

Recent proposals in this direction include generic algorithms for computing the shortest path, the minimal paths and the disjoint paths. However, a thorough search of the relevant literature did not yield any generic algorithm for computing the K-shortest paths, which is fundamental to design routing schemes that provide path diversity and fault-tolerance. Thereby the state of the art on routing schemes includes deterministic proposals and just only one fault-tolerant proposal, which does not provide minimal routing and does not guarantee packet delivery.

This Thesis focuses on the problem of generic routing in CGs. The problem is analyzed from the Automatic Group Theory (AGT). The fundamental idea of the AGT is that the structure of every finite (and some infinite) CG can be encoded in a set of Finite State Automata (FSA) called *ShortLex* Automatic Structure (SAS). For a CG with diameter D_Γ, it has been proved that the shortest path problem can be solved in time $O(D_\Gamma^2)$ using its related SAS.

In this research work, word-processing techniques are used together with SASs to design low complexity algorithms for computing: the shortest path, the minimal paths, the paths of bounded length, the K-shortest paths, the disjoint paths, and the shortest path avoiding a set of nodes and links. Based on these algorithms, it is proposed a generic routing scheme that has low time and space complexity; guarantees packet delivery; and provides minimal routing, path diversity and fault-tolerance.

The routing scheme and path computation algorithms proposed are evaluated through a complexity analysis and a comparison with the state

of the art on generic routing for CGs. The contributions of this Thesis also include a theoretical framework to study and solve problems related to path computation and routing in CGs from an approach of word processing, and an analysis of the topological properties of CGs and their impact on the performance and robustness of networks that use them as topology.

RESUMEN

Los grafos de Cayley (CG, por sus siglas en inglés) son una representación geométrica de grupos algebraicos. Estos grafos han sido utilizados como topologías de una gran variedad de redes de comunicaciones, tales como redes de interconexión de procesadores, redes inalámbricas de sensores, redes de centros de datos, etc. El motivo es que sus propiedades de transitividad de nodo, conectividad de enlace y baja distancia promedio entre nodos hacen posible un alto desempeño y robustez en redes de gran escala.

La mayoría de la redes de comunicación, cuyas topologías son definidas por CGs, utilizan esquemas de encaminamiento tradicionales, tales como el *Encaminamiento Valiant* y el *Encaminamiento Universal, Globalmente Adaptativo y con Balance de Carga*. Estos esquemas de encaminamiento emplean algoritmos de búsqueda de caminos que son agnósticos a la topología, tales como los algoritmos de Dijkstra y Bellman-Ford. Estos algoritmos reciben todo el grafo como entrada, lo que resulta en una alta complejidad de tiempo y espacio para los esquemas de encaminamiento que los utilizan.

En contraste, se han diseñado esquemas de encaminamiento y algoritmos de búsqueda de caminos especificos para CGs, con el objetivo de lograr baja complejidad espacial y temporal mediante el aprovechamiento de las (antes mencionadas) propiedades topológicas de los CGs. Estas propuestas pueden ser clasificadas en: 1) genéricas, es decir que funcionan en varias familias de CGs; y 2) especializadas en una familia de CGs. El reto de lograr baja complejidad espacial y temporal es mayor en el caso de propuestas genéricas, debido a que estas propuestas deben funcionar en CGs con estructuras topológicas diferentes.

Trabajos recientes en esta dirección incluyen algoritmos genéricos para la búsqueda de: el camino más corto, los caminos mínimos y los caminos disjuntos. Sin embargo, después de una búsqueda exhaustiva de la literatura relevante, no se encontró ningún algoritmo genérico para la búsqueda de los K caminos más cortos, lo cual es fundamental para el diseño de esquemas de encaminamiento que provean diversidad de caminos y tolerancia a fallos. En consecuencia, los de esquemas de encaminamiento más modernos para CGs incluyen propuestas deterministas y sólo una propuesta tolerante a fallos, la cual no provee encaminamiento mínimo ni tampoco garantiza la entrega de paquetes.

Esta Tesis se enfoca en el problema de encaminamiento genérico en CGs. El problema es analizado desde el punto de vista de la *Teoría de Grupos Automáticos* (AGT, por sus siglas en inglés). La idea fundamental de la AGT es que la estructura de todo CG finito (y algunos infinitos) puede ser codificada en un conjunto de *Autómatas de Estados Finitos*, dicho conjunto es llamado *Estructura Automática Lexicográficamente más Corta* (SAS, por sus siglas en inglés). Para un CG con diámetro D_Γ se ha probado que el problema de encontrar el camino más corto puede ser resuelto en tiempo $O(D_\Gamma^2)$ si se utiliza su respectiva SAS.

En este trabajo de investigación se utilizan técnicas de procesamiento de texto junto con SASs para diseñar algoritmos de baja complejidad que encuentran: el camino más corto, los caminos mínimos, los caminos de longitud limitada, los K caminos más cortos, los caminos disjuntos, y el camino más corto que excluye un conjunto de nodos y enlaces. Con base en estos algoritmos se ha propuesto un esquema de encaminamiento genérico para CGs, el cual tiene baja complejidad espacial y temporal, garantiza la entrega de paquetes, y provee: encaminamiento mínimo, diversidad de caminos y tolerancia a fallos.

Tanto el esquema de encaminamiento como los algoritmos de búsqueda de caminos propuestos son evaluados mediante un análisis de complejidad y una comparación con las propuestas más modernas de encaminamiento genérico para CGs. Las contribuciones de esta Tesis también incluyen un marco teórico para estudiar y resolver problemas relacionados con la búsqueda de caminos y encaminamiento en CGs desde un enfoque de procesamiento de texto, y un análisis de las propiedades topológicas de los CGs y su impacto en el desempeño y la robustez de las redes que los utilizan como topología.

RESUM

Els grafs de Cayley (CG, per les seves sigles en anglès) són una representació geomètrica de grups algebraics. Aquests grafs han estat utilitzats com topologies d'una gran varietat de xarxes de comunicacions, com ara xarxes d'interconnexió de processadors, xarxes inalàmbriques de sensors, xarxes de centres de dades, etc. Això és principalment per les seves propietats de transitivitat de node, connectivitat d'enllaç i baixa distància mitjana entre nodes que fan que xarxes de mides molt grans tinguin un bon rendiment i siguin robustes.

La majoria de xarxes de, comunicació amb topologies definides per CGs, utilitzen esquemes d'encaminament tradicionals, com ara l' Encaminament Valiant i l'Encaminament Universal, Globalment A-daptatiu i amb Balanç de Càrrega. Aquests esquemes d'encaminament utilitzen algoritmes de cerca de camins que són agnòstics a la topologia com ara els algoritmes de Dijkstra i Bellman-Ford. Aquests reben tot el graf com a entrada, el que resulta en una alta complexitat en temps i espai per als esquemes d'encaminament que els utilitzen.

Per contra, s'han dissenyat esquemes d'encaminament i algoritmes de cerca de camins específics per a CGs, amb l'objectiu d'aconseguir baixa complexitat de espai i temporal mitjançant l'aprofitament de les (abans esmentades) propietats topològiques dels CGs. Aquestes propostes es poden classificar en: 1) genèriques, és a dir, que funcionen en diverses famílies de CGs; i 2) especialitzades en una família de CGs. El repte d'aconseguir baixa complexitat de espai i temporal és més gran en el cas de propostes genèriques, atès que aquestes propostes han de funcionar en CGs amb estructures topològiques diferents.

Treballs recents en aquesta direcció inclouen algoritmes genèrics per a la cerca de: el camí més curt, els camins mínims i els camins disjunts. No obstant, després d'una recerca exhaustiva de la literatura rellevant, no es va trobar cap algoritme genèric per a la recerca dels K camins més curts, la qual cosa és fonamental per al disseny d'esquemes d'encaminament que proveeixin diversitat de camins i tolerància a fallades. En conseqüència, els esquemes d'encaminament més moderns inclouen propostes deterministes i tan sols una proposta tolerant a fallades, la qual no proveeix encaminament mínim ni tampoc garanteix el lliurament de paquets.

Aquesta Tesi està enfocada al problema d'encaminament genèric en CGs. El problema és analitzat des del punt de vista de la Teoria de Grups Automàtics (AGT, per les seves sigles en anglès). La idea fonamental de la AGT és que l'estructura de tot CG finit (i alguns infinits) pot ser codificada en un conjunt d'Autòmats d'Estats Finits, aquest conjunt és anomenat Estructura Automàtica Lexicogràficament més Curta (SAS, per les seves sigles en anglès). Per a un CG amb diàmetre D_Γ s'ha provat que el problema de trobar el camí més curt pot ser resolt en temps $O(D_\Gamma)$ si s'utilitza la seva respectiva SAS.

En aquest treball d'investigació s'utilitzen tècniques de processament de text junt amb SASs per a dissenyar algoritmes de baixa complexitat que

trobin: el camí més curt, els camins mínims, els camins de longitud limitada, els K camins més curts, els camins disjunts, i el camí més curt que exclou un conjunt de nodes i enllaços. Amb base a aquests algoritmes s'ha proposat un esquema d'encaminament genèric per a CGs que té baixa complexitat de espai i temporal, garanteix el lliurament de paquets, i proveeix: encaminament mínim, diversitat de camins i tolerància a fallades.

Tant l'esquema d'encaminament com els algoritmes de cerca de camins propossats són avaluats mitjançant una anàlisi de complexitat i una comparació amb les propostes més modernes en encaminament genèric per a CGs. Les contribucions d'aquesta Tesi també inclouen un marc teòric per estudiar i resoldre problemes relacionats amb la cerca de camins i encaminament en CGs, des d'un enfocament de processament de text, i una anàlisi de les propietats topològiques dels CGs i el seu impacte en el rendiment i la robustesa de xarxes que els utilitzen com a topologia.

It is not only the question,
but the way you try to solve it.

— Maryam Mirzakhani

1 INTRODUCTION

1.1 MOTIVATION

Since S. Akers introduced a group theoretic model for interconnection networks [1], Cayley Graphs (CGs) have been used as topologies of a wide variety of communication networks [2–6]. The reason is that the topological properties of CGs enable high performance and robustness in large-scale networks [7, 8]. Before proceeding to examine the topological properties of CGs, it is necessary to give the formal definition of these graphs.

Let $\mathcal{G}$ be an algebraic group with a generating set S [9, Section 2.2]. Then $\mathcal{G}$ has an associated CG, denoted by $\Gamma(\mathcal{G}, S)$. The set of vertices is given by the set of group elements. Let $g, h \in \mathcal{G}$, there is an edge from g to h if and only if $g \cdot s = h$ for some $s \in S$. Cayley graphs can be classified into families according to its related group and generating set. For instance, consider the additive group $\mathbb{Z}_p^2 = \{(x, y) \in \mathbb{Z}^+ \times \mathbb{Z}^+ : 0 \leqslant x, y < p\}$ with generating set $S = \{(1, 0), (0, 1)\}$. The group operation is the addition module p. A member of the family of CGs of the additive group $\mathbb{Z}_p^2$ with generating set $S = \{(1, 0), (0, 1)\}$ is shown in Figure 1. Note that CGs are regular graphs, i.e. nodes have the same number of edges.

The three key topological properties of CGs that enable high performance and robustness in communication networks are:

1) **Node-transitivity**. Roughly speaking, all the nodes of a node-transitive graph have the same perspective of the whole graph. Then nodes can not been distinguished each other with respect to their position in the network, which allows the design of simple communication protocols.

2) **Link-connectivity**. It is a consequence of the node-transitivity and refers to the minimal number of links that must be removed to disconnect the network. Then high link-connectivity allows fault-tolerance for random link failures. Cayley graphs have the maximum possible value of link-connectivity for regular graphs. In addition, several CGs have also the maximum possible value of node-connectivity [7].

3) **Low average distance**. An important issue in the design of network topologies is to connect the maximum number of nodes while keeping low average distance between them and thus low latency. For regular networks, the well-known *Moore bound* states the minimum value of average distance for a given number of nodes [10, Theorem 10.1]. Several CGs used as topologies of communication networks satisfy such bound [7].

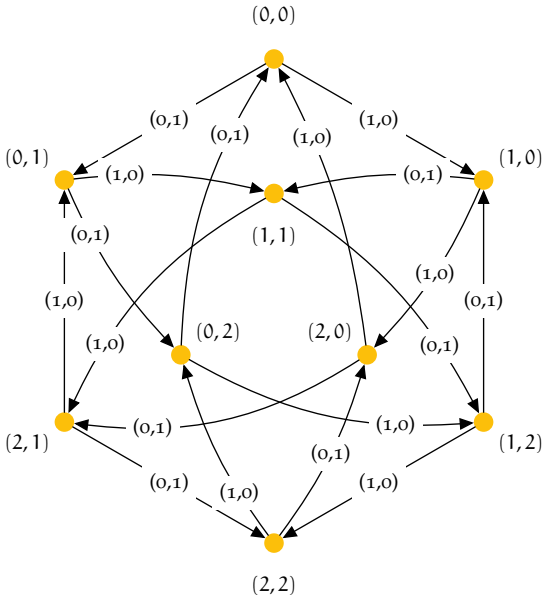


Figure 1: Cayley graph of the additive group $\mathbb{Z}_3^2$ with generating set $S = \{(1,0),(0,1)\}$.

Most of communication networks, whose topology is a CG, apply traditional routing schemes [11, 12] that apply topology-agnostic algorithms for path computation [13, 14]. It results in high space and time complexity due to such algorithms take the whole graph as input. In contrast, routing proposals dedicated to CGs can be designed to achieve low memory and time requirements taking advantage of the topological properties of CGs. These proposals can be classified into: 1) generic, and 2) specialized on a family of CGs. The challenge of achieving low time and space requirements is major for generic proposals as they must work on CGs with different topological structure.

Recent generic proposals of routing in CGs include deterministic routing schemes [1, 15] and a fault-tolerant routing scheme [16], which does not guarantee packet delivery. In addition, algorithms for computing the minimal paths [17] and the vertex-disjoint [18] has been proposed. However, a thorough search of the relevant literature did not yield any generic algorithm for computing the K-shortest paths, which is fundamental to design routing schemes that provide path diversity and fault-tolerance. This Thesis focuses on the problem of generic routing in CG with the aim of designing a routing scheme that guarantees packet delivery and provides path diversity and fault-tolerance by applying path computation algorithms with low time and space requirements.

1.2 PROBLEM STATEMENT

This section presents basic definitions of routing in general and routing in CGs. Further details about network and routing models can be found in [19].

1.2.1 *The network model*

Consider a point-to-point communication network, its arrangement of nodes and links is called topology. A network topology can be described by a graph $G = (V, E)$, where vertices $V(G)$ represent the nodes of the network and edges $E(G)$ the links between nodes. Each node $u \in V(G)$ has $deg(u)$ ports numbered from 1 to $deg(u)$, which connect the links of u as it is shown in Figure 2.

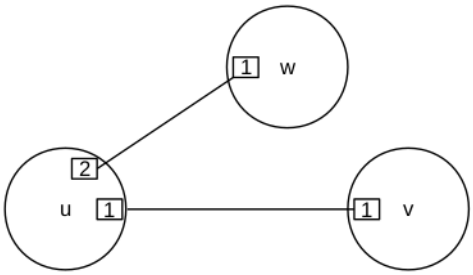


Figure 2: The node u and the links (u, w) and (u, v) that connect it to nodes w and v, respectively. In undirected graphs, $(u, w) = (w, u)$ for every $(u, w) \in E(G)$

.

1.2.2 *Routing basics*

1.2.2.1 *The routing scheme model*

The routing process consists in finding a path between a (source) node to another (destination) node in order to deliver messages from the source node to the destination node. In this Thesis, it is considered the routing method called *store and forward*, which forwards messages from the source node to the destination node through a chain of intermediate nodes. Each of them stores the message and decides through which link each message is forwarded.

In general, a routing scheme consists of:

1) **Routing information**. It can be routing tables that assign output ports to destination nodes, and/or data structures storing details of the state or structure of the network.

2) **Forwarding protocols**. They are mechanisms responsible for delivering and forwarding messages. They uses the routing information and algorithms to compute the message headers and to decide which is next node in the path.

The forwarding protocols work in source and intermediate nodes as follows. First, a routing scheme requires that nodes had been labeled with a unique identifier (label). Suppose that a source node v wants to deliver a message `MSG` to a destination node u. The forwarding protocol must execute the following steps:

1) Compute a header and attach it to `MSG`.

2) Decide the exit port and forward `MSG` to it.

Now, consider an intermediate node along the path of `MSG`. The forwarding protocol must perform the following steps:

1) Decide if the `MSG` has arrived to its final destination. If so, the forwarding protocol finishes. Otherwise:
   - Compute a header and attach it to `MSG`.
   - Decide the exit port and forward `MSG` to it.

1.2.2.2 *Objectives of routing schemes*

The objectives of routing schemes depend on the kind of the communication network (e.g. Processor Interconnection Networks (PINs), Wireless Sensor Networks (WSNs), Data Center Networks (DCNs), etc.) and its application (e.g. storage, High Performance Computing (HPC), etc.). One or more of the following objectives can be considered as primaries for several communication networks:

- **Path diversity**. Exploit the path diversity of the network topology, which include both minimal and non-minimal paths.
- **Load balancing**. Distribute the traffic between links in order to avoid congestion and achieve high throughput.
- **Fault-tolerance**. Guarantee packet delivery in spite of failures of nodes and links.
- **Complexity efficiency**. The forwarding protocols must be implemented efficiently, i.e. they must have low time and space requirements.

1.2.2.3 *Features of routing schemes*

Routing schemes can be classified according to the following features [20, Chapter 5]:

- Topology:
  - **Topology-agnostic**. The routing scheme has no assumptions about the topology structure and then it can work on any topology.
  - **Specialized**. The routing scheme only works on specific topologies.
- Adaptability:
  - **Adaptive routing**. The routing paths are computed taking into account the state of the network, e.g. congested or failed nodes and links.
  - **Deterministic routing**. No network state is taken into account to compute the routing paths. Then, the routing paths are always the same for a given source and destination nodes.
- Hop count:

  - **Minimal routing**. Messages are routed by the minimal paths. Depending on the network topology, there might be multiple minimal paths for a given pair of nodes.
  - **Non-minimal routing**. Messages are not always sent by minimal paths. Non-minimal routing allows adaptive routing taking advantage of the path diversity.

- Routing decision:
  - **Source routing**. The routing path is computed just once at the source node.
  - **Hop-by-hop routing**. Each node in the path takes a routing decision based on the destination label and the routing information. Then, the routing path is built along all nodes in the path.

- Routing implementation:
  - **Algorithmic**. Based on the routing information and destination label, algorithms are used to compute the message header and output port.
  - **Table-based routing**. The routing information consists of a look-up routing table that might assign the output port for a given destination node.

1.2.2.4 *Path computation algorithms*

Path computation is a fundamental task of routing schemes. In general, algorithms can be designed to compute the following paths between two nodes: 1) the shortest path, 2) the K-shortest paths, 3) the disjoint paths, i.e. paths that do not share nodes and/or links. Taking these algorithms as base, new algorithms can be designed, in order to compute: the minimal paths, the paths of bounded length, etc.[1] The objectives reached by a routing scheme depend on the kind of path computation algorithms applied and their implementation. In fact, the major challenge in the design of routing schemes may consists in applying (or designing) the proper path computation algorithms. For instance, path diversity and fault-tolerance are provided by computing the K-shortest path; load balancing is provided by computing the disjoint paths; complexity effectiveness highly depends on the complexity of the path computation algorithms.

1.2.3 *Complexity measures*

Traditional complexity measures can be used to evaluate the performance of both path computation algorithms and routing schemes [19, Chapter 2]. Regarding path computation algorithms, the common complexity measures are:

1 It is important to distinguish between the minimal paths and the K-shortest paths. A path that has minimal length is called minimal path. For instance, if the 3-shortest paths between two nodes have length 4, 4 and 6, respectively, then the paths with length 4 are minimal.

- **Space complexity**. The amount of memory in bits used by the algorithm in the worst case.
- **Time complexity**. The number of time units that the algorithm takes to finish in the worst case.

Regarding routing schemes in the *asynchronous communication model*[2] [19, Section 1.3.4], the common complexity measures are:

- **Memory space requirements per node**. The amount of memory in bits used by a node, in the worst case, to store its label, routing table and forwarding algorithm.
- **Forwarding decision time**. The number of time units that the forwarding algorithm takes to find the next node in the path in the worst case.
- **Packet delivery time**. The number of time units that a packet takes to reach its destination node from its source node in the worst case.
- **Complexity of distributed processes**. The majority of routing schemes employ distributed processes to communicate the states of the network, such as initial configuration or notification about any failure. These processes involve the exchange of messages among the nodes of the network. The common complexity measures of distributed processes are:
  - **Convergence time**. The number of time units from the beginning of the execution of the process to its completion in the worst case and assuming that each message incurs a delay of at most one time unit.
  - **Message complexity**. The total number of basic messages transmitted during the execution of the process in worst case.

1.2.4 *Generic routing in Cayley graphs*

Generic routing proposals for CGs usually follow a similar strategy that consists in obtaining a generic representation of different families of CGs, and then the routing problem is solved on such representation. These representations includes: permutations [21, 22] *chordal rings* graphs [23], Finite State Automata (FSA) [24], etc.

This Thesis employs a representation of CGs as FSA, such approach arises from the Automatic Group Theory (AGT). The fundamental idea of the AGT states that CGs can be described by a linear recursion. Hence every CG can be constructed from repetitions of a finite subgraph of it.A formal definition of this linear recursion is given through FSA that encode the structure of their related CG. The set of FSA related to a CG is called *ShortLex* Automatic Structure (SAS)[3].

2 In this communication model, there is no global clock signal and thus a message sent from a node to one of its adjacent nodes arrive within some finite but unpredictable time. Therefore, algorithms are event-driven.

3 Although all finite CG has an associated SAS, not all infinite CG has it.

From the SAS of a CG, low complexity algorithms can be constructed. For instance, it has been proved that the shortest path problem can be solved in time $O(D_\Gamma^2)$, where D_Γ is the diameter of the CG [25, Theorem 2.3.10]. This result was applied in [15] to design a deterministic routing scheme with low time and space complexity. This Thesis extends and enhances such work applying techniques of word processing to design a set of path computation algorithms with low time and space requirements. Then, these algorithms are used to support a routing scheme that provides fault-tolerance.

1.3 OBJECTIVES

This Thesis studies the routing problem in networks whose topology is given by a CG. Its overall objective can be stated as follows:

To design a generic routing scheme for CGs that has low time and space requirements, guarantees packet delivery, and provides: minimal routing, path diversity and fault-tolerance.

This objective can be divided into the following specific objectives:

- To conduct a review of generic routing proposals for CGs. It includes path computation algorithms and routing schemes.
- To design a theoretical framework to study and solve problems related to path computation and routing in CGs from an approach of AGT.
- To design a set of generic algorithms for path computation in CGs that have low time and space complexity.

1.4 CONTRIBUTIONS

The main contributions of this research work are:

- An analysis of the topological properties of CGs and their impact on the performance and robustness of networks that used them as topology. It includes a comparison of a set of specific families of CGs.
- An state of the art on generic routing schemes and path computation algorithms for CGs. It includes a complexity analysis and evaluation of their time and space requirements on specific families of CGs.
- A theoretical framework to study and solve problems related to path computation and routing in CGs from an approach of AGT.
- A set of generic algorithms for path computation in CGs. Specifically, the proposed algorithms compute:
  1. the shortest path,
  2. the minimal paths,
  3. the paths of bounded length,

4. the K-shortest paths,
5. the disjoint paths, and
6. the shortest path avoiding a set of nodes and edges.

These algorithms work on any finite CG and several infinite CGs.

- A generic routing scheme for CGs that guarantees packet delivery and provides: minimal routing, path diversity and fault-tolerance.

1.5 OUTLINE OF THE DOCUMENT

This document is divided into two parts. Part I analyzes the use of CGs as topologies of communication networks, and the routing problem in such networks. Part II focuses on the main contribution of this Thesis that is a routing scheme for CGs called Word-Processing-based Routing (WPR).

Figure 3 summarizes the dependencies among the chapters of this document and the relations between chapters and the Publications resulting from this Thesis. Chapter 1, Chapter 6 and Chapter 11 are recommended to have an overview of this research work. Part II is necessary to obtain a depth understanding of the WPR.

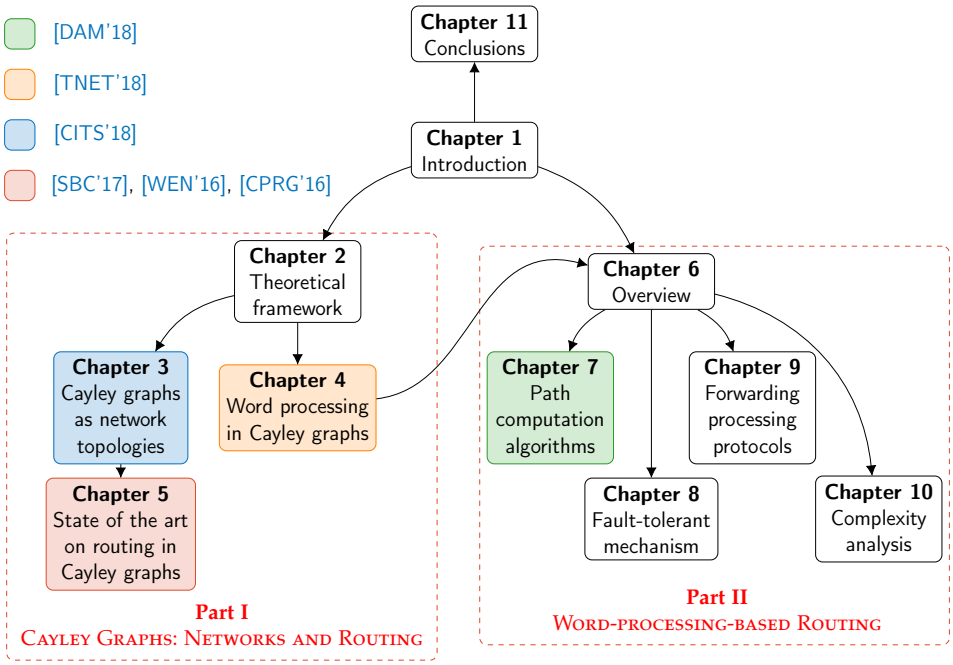


Figure 3: Dependency among chapters and relation to publications.

1.5.1 *Part I. Cayley graphs: networks and routing*

Chapter 2 introduces the terminology of graph theory, regular languages and group theory that supports this research work. Chapter 3 analyzes the impact of the topological properties of CGs on the performance and robustness of networks that use them as topology. This analysis contributes to a better understanding of how routing proposals can be designed taking advantage of the topological properties of CGs. In addition, the properties

of six well-known families of CGs are analyzed and evaluated. These results are used in Chapter 5 and Chapter 10 to evaluate the complexity of the state of the art on routing proposals and the WPR, respectively. Chapter 4 explains the word processing approach on which the WPR is based. This approach arises from the AGT. Chapter 5 conducts the state of the art on generic routing in CGs, which has been divided into path computation algorithms and routing schemes. This chapter also includes a complexity analysis of the routing proposals and an evaluation of their time and space complexity on a set of specific families of CG.

1.5.2 *Part II. Word-Processing-based Routing*

Chapter 6 gives an overview of the operation of WPR, its features and the the process of node label assignment. Chapter 7 presents the algorithms for path computation used by the WPR. Chapter 8 explains the operation of the fault-tolerant mechanism for both node and link failures. Chapter 9 introduces the forwarding protocols for deterministic and fault-tolerant routing. For deterministic routing, forwarding protocols for single and multi-path routing are presented. Meanwhile, for fault-tolerant routing, only a forwarding protocol for single-path routing is presented. Chapter 10 presents a complexity analysis of the WPR and their path computation algorithms. It is also included an evaluation of the space and time requirements on a set of families of CG together with a comparison with the state of the art on routing proposal for CGs. Finally, Chapter 11 discusses the concluding remarks and directions for further research.

Part I

CAYLEY GRAPHS: NETWORKS AND ROUTING

Cayley graphs provide a geometric representation of algebraic groups. These graphs have been used in a wide variety of communication networks as their topological properties enable high performance and robustness. The following chapters analyze the use of Cayley graphs as topologies of communication networks, and the routing problem in such networks.

The mathematician's patterns,
like the painter's or the poet's must be beautiful;
the ideas like the colours or the words,
must fit together in a harmonious way.

— Godfrey H. Hardy

2 THEORETICAL FRAMEWORK

This Chapter introduces the terminology of graph theory, regular languages and group theory that supports the research presented in this Thesis.

2.1 GRAPH THEORY

This section introduces some basic notation and graph-theoretic terminology that is frequently used in the description of routing in networks. Further information about graph theory and its application in the study of routing in networks can be found in [19, 21, 26].

2.1.1 *Graphs and subgraphs*

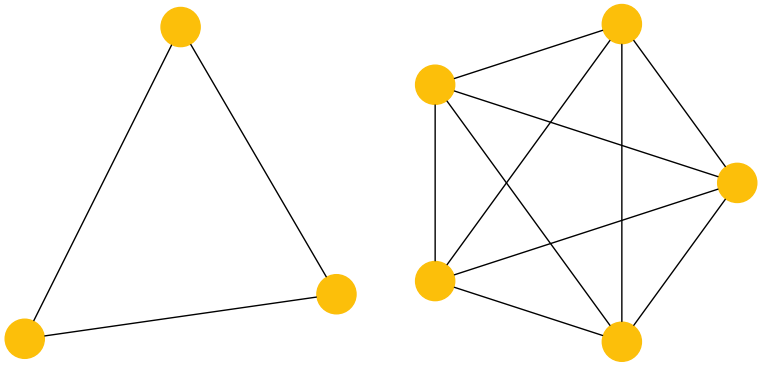
Definition 2.1.1 *A **graph**, denoted by $G = (V, E)$, is an object consisting of two sets called its **vertex set** $V(G)$ and its **edge set** $E(G)$. The elements of $V(G)$ are called **vertices** and the elements of $E(G)$ are called **edges**. The number of vertices are denoted by $n = |V(G)|$ and the number of edges by $m = |E(G)|$.*

Let $(u, v) \in E(G)$ be an edge of G, then it is said that (u, v) joins the vertices $u, v \in V(G)$ and that u and v are **adjacent**. The edge (u, v) is **incident** to each u and v, and each u and v is **incident** to (u, v). Two edges incident to the same vertex are called **incident** edges. A vertex incident to no edges is called **isolated vertex**.

Definition 2.1.2 *The **degree of a vertex** v, denoted by $\deg(v)$, is the number of edges incident to it. The **degree of a graph** $G = (V, E)$, denoted by Δ_G, is the maximum degree of a vertex, i.e. $\Delta_G = \max\{\deg(v) : v \in V(G)\}$.*

Graphs are often illustrated by pictures on the plane. The vertices $V(G)$ are represented by points and the edges by lines connecting adjacent vertices. Figure 4 show examples of the following graphs:

- A **Δ_G-regular graph**, where every vertex has degree Δ_G.
- A **complete graph**, denoted by K_n, which has n vertices, such that every pair of vertices in $V(K_n)$ are adjacent. Note that K_n is $(n-1)$-regular.



K_3 is a 2-regular graph K_5 is a 4-regular graph

Figure 4: Examples of graphs

Definition 2.1.3 *Let $G = (V, E)$ be a graph, and let V' and E' be subsets of V and E, respectively. The graph $G' = (V', E')$ is called a **subgraph** of G. The graph G' is called **induced subgraph** of G if E' is given by all edges $e \in E$ whose incident vertices are in V'. The graph G' is an **spanning subgraph** of G if $V' = V$.*

2.1.2 *Paths, connectedness and trees*

Definition 2.1.4 *A sequence of edges $e_1, \ldots, e_l$ joining two vertices $v_0, v_l \in V(G)$ is called a **walk**. If $v_0 = v_l$, then it is called a **closed walk**. If all edges e_i in a walk are distinct, then it is called a **trail**.*

Definition 2.1.5 *Let $v_0, \ldots v_l$ a sequence of vertices such $e_i = (v_{i-1}, v_i)$ for every e_i in a trail. If every v_j are distinct, then the trail is called a **path**. A closed path for any $l > 2$ is called a **cycle**.*

Definition 2.1.6 *The **length of a path** is equal to its number of edges.*

A path between two vertices $V(G)$ is called **minimal** if its length is minimal, i.e. there is no path between the same two nodes that is shorter that it. Two o more paths between any two vertices in $V(G)$ are called **edge-disjoint** if all their edges are distinct. In addition, they are called **vertex-disjoint** if all their vertices are distinct except the first and last vertices.

Definition 2.1.7 *A graph G is called **connected** if there exists a path between any two vertices of G. A **tree**, denoted by T, is a connected graph such that the following definitions are equivalent:*

(i) *T has no cycles.*

(ii) *There is only one path between any two vertices in $V(T)$.*

2.1.3 *Weighted graphs, distances and neighborhoods*

Definition 2.1.8 *Let $G = (\omega, V, E)$, where $\omega : E(G) \to \mathbb{R}^+$ is a function that assigns a numerical weight to each edge $e \in E(G)$. Then, G is called **weighted graph**. For unweighted graphs, it is assumed that all edges have weight equal to 1.*

Definition 2.1.9 *Let* $d_G(u,v)$ *denote the* ***distance*** *between any two vertices* u, v *of a unweighted graph* G*, which is the length of the the shortest path between them.*

Definition 2.1.10 *Let* D_G *denote the* ***diameter of a graph*** $G = (V, E)$*, given by the maximal distance between any two vertices in it, i.e.* $D_G = \max\{d_G(u,v) : u, v \in V(G)\}$.

Definition 2.1.11 *The* ***neighborhood of a vertex*** $v \in V(G)$*, denoted by* $N(v)$*, is the set of its neighbors, i.e. the vertices adjacent to it, including* v*. The* l***-neighborhood of a vertex*** $v \in V(G)$*, denoted by* $N(v,l)$*, is the set of vertices at distance at most* l *from* v*, i.e.* $N(v,l) = \{u \in V(G) : d_G(u,v) \leqslant l\}$.

2.1.4 *Graph isomorphism*

Definition 2.1.12 *Let* $G = (V, E)$ *and* $G' = (V', E')$ *be two graphs. Let* $\pi : V(G) \to V'(G')$ *be a bijective map such that for all* $(u,v) \in E(G)$*, such that if* $(u,v) \in E(G)$*, then* $(\pi(u), \pi(v)) \in E'(G')$*. Thereby* π *defines a* ***graph isomorphism*** *and* G *and* G' *are said to be* ***isomorphic****, i.e.* $G \simeq G'$

Definition 2.1.13 *Let* $\pi : V(G) \to V'(G')$ *be a graph isomorphism such that* $G = G'$*, then* π *defines an* ***automorphism*** *of* G.

2.2 GROUP THEORY

2.2.1 *Groups and subgroups*

Definition 2.2.1 *Let* $(\mathcal{G}, \circ)$ *be an ordered pair, where* $\mathcal{G}$ *is a non-empty set and* $\circ$ *is a binary operation on* $\mathcal{G}$*. Consider now the following axioms for all* $f, g, h \in \mathcal{G}$*:*

(i) Closure

$$g \circ h \in \mathcal{G}.$$

(ii) Associativity

$$(f \circ g) \circ h = f \circ (g \circ h).$$

(iii) Identity element

$$\exists \epsilon \in \mathcal{G}, \text{ such that } g \circ \epsilon = \epsilon \circ g = g.$$

(iv) Inverse element

$$\exists g^{-1} \in \mathcal{G}, \text{ such that } g \circ g^{-1} = \epsilon.$$

(v) Commutativity

$$g \circ h = h \circ g.$$

Then:

- *If axioms (i) and (ii) hold, then* $(\mathcal{G}, \circ)$ *is a* ***semigroup****.*
- *If axioms (i), (ii) and (iii) hold, then* $(\mathcal{G}, \circ)$ *is a* ***monoid****.*
- *If axioms (i), (ii), (iii) and (iv) hold, then* $(\mathcal{G}, \circ)$ *is a* ***group****.*

- *If axioms (i), (ii), (iii), (iv) and (v) hold, then* $(G, \circ)$ *is an* ***abelian group****.*

To simplify notation, G *denotes* $(G, \circ)$ *and* gh *denotes* $g \circ h$*, where* $g, h \in G$*. The cardinality of* G*, i.e.* $|G|$*, is called the* ***order*** *of* G*.*

Definition 2.2.2 *A subset* H *of a group* G *is called* ***subgroup*** *of* G*, i.e.* $H < G$*, if it forms a group under the same group operation as* G*.*

2.2.2 *Homomorphism*

Definition 2.2.3 *Let* $(G, \circ)$ *and* $(H, *)$ *be two groups and let* $\pi : G \to H$ *be a map satisfying the following condition* $\pi(g \circ h) = \pi(g) * \pi(h)$*, for all* $g, h \in G$*. Then* π *defines an* ***homomorphism*** *from* G *to* H*, such that:*

- *If* π *is surjective, then* π *is an* ***epimorphism****.*
- *If* π *is injective, then* π *is a* ***monomorphism****.*
- *If* π *is bijective, then* π *is an* ***isomorphism****.*

Definition 2.2.4 *Let* $\pi : G \to H$ *be a homomorphism, there are two important subgroups related to* π*:*

$$Ker\ \pi = \{g \in G : \pi(g) = \epsilon\}$$

and

$$Im\ \pi = \{h \in H : h = \pi(g), \text{ such that } g \in G\}.$$

Then Ker π *is a subgroup of* G *and is called* ***kernel*** *of* π*; Im* π *is a subgroup of* H *and is called* ***image*** *of* π*.*

2.2.3 *Group presentations*

Definition 2.2.5 *Let* S *be a subset of a group* G*. The subgroup of* G *generated by* S *is given by the intersection of all subgroups of* G *that contain* S*, such a subgroup is denoted by* $\langle S \rangle$*. It is said that* S *generates* G *(or is a* ***generating set*** *for* G*) if* $\langle S \rangle = G$*.*

Definition 2.2.6 *A group* F *is free over the subset* $A \subset F$ *if for any group* G *and any map* $\alpha : A \to G$*, there is a unique group homomorphism, more concretely an epimorphism,* $\pi : F \to G$*. The* ***free group*** *on a set* A *is denoted by* $F(A)$*, where* A *generates* $F(A)$*.*

Definition 2.2.7 *Let* π *as was defined in Definition 2.2.6. Then,* π *defines a* ***free presentation*** *of* G*. Note that* $S = \{s : s = \pi(a), \forall a \in A\}$ *generates* G*, i.e.* $G = \langle S \rangle$*. In addition, if* $Y = Ker\ \pi$ *and* $R = \pi(Y)$*, then* R *is called the* ***set of relators*** *of* G*. Both sets* S *and* R *define a* ***presentation*** *of* G*, denoted by* $G = \langle S|R \rangle$*. If* S *is finite, it is said that* $G = \langle S|R \rangle$ *is a* ***finitely presented group****.*

2.2.4 *Permutation Groups*

Definition 2.2.8 *Let* A *be a finite set of elements, a* ***permutation*** *of* A *is a bijection* $\sigma : X \to X$*. The set of all permutations of* A *is denoted by* S_X*. Hereafter it is assumed that* $X = \{1, 2, \ldots, p\}$*, then* S_X *is denoted by* S_p*.*

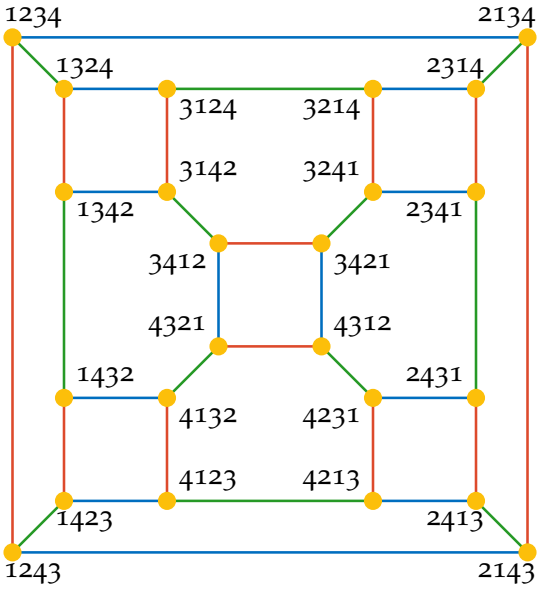


Figure 5: Cayley graph of the symmetric group S_4 with generators $(2,1,3,4)$, $(1,3,2,4)$, $(1,2,4,3)$ represented by blue, green and red edges, respectively. This Cayley graph is known as *Bubble-sort* graph.

A permutation $\sigma \in S_p$ can be viewed as a rearrangement of all elements from $X = \{1, 2, \ldots, p\}$, which can be written in **two-line notation**, i.e.

$$\sigma = \begin{pmatrix} 1 & 2 & \ldots & p \\ \sigma(1) & \sigma(2) & \ldots & \sigma(p) \end{pmatrix},$$

where the bottom row is a rearrangement of $\{1, 2, \ldots, p\}$. A more compact notation is the **cycle notation**.

Definition 2.2.9 *Let $i \in X$, then σ **fixes** i if $\sigma(i) = i$ and **moves** i if $\sigma(i) \neq i$. Let $i_1, i_2, \ldots, i_r$ be distinct integers from X. If $\sigma(i_1) = i_2$, $\sigma(i_2) = i_3$, ..., $\sigma(i_{r-1}) = i_r$, $\sigma(i_r) = i_1$ and σ fixes the remaining $p - r$ integers from A, then σ can be written in **cycle notation** as $(i_1 i_2 \ldots i_r)$.*

Note that two permutations $\sigma_1, \sigma_2 \in S_p$ can be *multiplied* under composition. For example, if $\sigma_1 = (13)$ and $\sigma_2 = (123)$ are permutations from S_3. The product $\sigma_1\sigma_2$ is computed as follow:

$$\sigma_1\sigma_2(1) = \sigma_1(\sigma_2(1)) = \sigma_1(2) = 2,$$

$$\sigma_1\sigma_2(2) = \sigma_1(\sigma_2(2)) = \sigma_1(3) = 1,$$

$$\sigma_1\sigma_2(3) = \sigma_1(\sigma_2(3)) = \sigma_1(1) = 3.$$

Then, $\sigma_1\sigma_2$ can be written in cycle notation as (12).

Definition 2.2.10 *A **permutation group** is a group whose elements are permutations of a given set X and the group operation is the composition of permutations.*

Definition 2.2.11 *The set of all permutations of a set X forms a group under the composition of permutations, which is called the **symmetric group** and denoted by S_p. This group has order $p!$ and its identity element is $(1 \ldots p)$.*

Corollary 2.2.1 *Every permutation group is a subgroup of the symmetric group.*

Theorem 2.2.1 (Cayley's theorem) *Every group is isomorphic to a subgroup of the symmetric group, i.e. to a permutation group [9, Theorem 1.6.8].*

From the Cayley's theorem, results that are true for permutation groups, are true for every group.

2.2.5 *Cayley graphs*

Definition 2.2.12 *The graph $\Gamma(\mathcal{G}, S)$ is called the CG of the group $\mathcal{G}$ with respect to its generating set S. Each group element in $\mathcal{G}$ corresponds to a node in $V(\Gamma)$, i.e. there is a bijective map*

$$\gamma : \mathcal{G} \to V(\Gamma), \tag{1}$$

and two nodes $u, v \in V(\Gamma)$, where $u = \gamma(g)$ and $v = \gamma(h)$, are adjacent if and only if g and h are elements of $\mathcal{G}$ that satisfy $g \circ s = h$ for some $s \in \{S \cup S^{-1}\}$, where S^{-1} is the set of inverses of $\mathcal{G}$.

Note that CGs have regular degree $|S \cup S^{-1}|$. For instance, consider the symmetric group S_p, the family of CGs defined by S_p and the generating set $S = \{(2, 1, 3, \ldots, p-1, p), (1, 3, 2, \ldots, p-1, p), \ldots, (1, 2, 3, \ldots, p, p-1)\}$ is called family of Bubble-sort graphs, i.e. $BS(p)$. Figure 5 shows the Bubble-sort graphs for S_4, i.e. $BS(4)$.

Euler's unintended message is very simple:
Graphs or networks have properties,
hidden in their construction,
that limit or enhance our ability
to do things with them.

— Albert-László Barabási

3

CAYLEY GRAPHS AS NETWORK TOPOLOGIES

3.1 TOPOLOGY MODEL

Consider a point-to-point communication network, its arrangement of nodes and links is called topology and can be defined by a connected graph $G = (V, G)$. The vertices $V(G)$ represent the nodes of the network, which could be routers, computers, etc. The edges $E(G)$ represent the link between nodes. Each node $v \in V(G)$ has $deg(v)$ (or Δ_G, in the case of regular graphs) ports numbered from 1 to $deg(v)$ and connected to links. Hereafter the terms vertices and nodes will be used interchangeably as well as the terms edges and links, and $\Gamma(\mathcal{G}, S)$ will be used instead of G to denote a network whose topology is defined by a CG. Table 1 presents the parameters of the network topologies used throughout this Thesis.

Table 1: Parameters of network topologies.

	Network topology $G = (V, E)$	**Network topology** $\Gamma(\mathcal{G}, S)$	**Definition**
Parameter	$V(G)$	$V(\Gamma)$	Nodes in the network
	$E(G)$	$E(\Gamma)$	Links between nodes.
	$n = \lvert V(G)\rvert$	$n = \lvert V(\Gamma)\rvert$	Number of nodes (order of G).
	$m = \lvert E(G)\rvert$	$m = \lvert E(\Gamma)\rvert$	Number of links (size of G).
	Δ_G	Δ_Γ	Network degree.
	D_G	D_Γ	Diameter.
	D_{avg}		Average path length.

3.2 TOPOLOGICAL PROPERTIES OF CAYLEY GRAPHS

This section presents an analysis of the topological properties of CGs and their impact on the performance and robustness of networks, whose topology is defined by CGs.

3.2.1 *Symmetry*

There are two properties of symmetry in graphs: node and link-transitivity. [7].

Definition 3.2.1 *A graph* G *is* ***node-transitive*** *if for every pair of nodes* $u, v \in V(G)$*, there is an automorphism* π *over* G *(Definition 2.1.13), such that* $u = \pi(v)$*.*

Roughly speaking, all the nodes of a node-transitive graph have the same perspective of the whole graph and thus can not be distinguished from any other based on their neighbourhoods (Definition 2.1.11). This property enables to perform distributed processes using the same algorithms in each node. For instance, distributed routing can be performed using the same routing algorithms in each node.

Definition 3.2.2 *A graph* G *is said to be* ***link-transitive*** *if for every pair of links* $d, e \in E(G)$*, there is an automorphism over links, i.e.* $\pi : E(G) \to E(G)$*, such that* $d = \pi(e)$*.*

In this case, all the links have the same perspective of the whole graph and thus can not be distinguished from any other based on the nodes and links surrounding it. Graphs that are both node and link-transitive are called **symmetric graphs**. Every CG is node-transitive but only some of them are symmetric [27, Section 2.1].

Theorem 3.2.1 *A group* $\mathcal{G}$ *is called* ***minimally generated*** *by a generating set* S*, if and only if it is generated by* S *and can not be generated by any other subset of* S*. Every CG of a minimally generated group is symmetric [28, Section 3.2].*

Theorem 3.2.2 *Cayley graphs with* $\Delta_\Gamma \leqslant 4$ *or* $\Delta_\Gamma = 6$ *are symmetric [29].*

3.2.2 *Connectivity and fault-tolerance*

There are two kinds of connectivity measures for graphs: node and link-connectivity.

Definition 3.2.3 *The* ***node-connectivity*** *of a graph* G *is the minimal number* c_n*, such that there are* c_n *node-disjoint paths between any pair of nodes in* $V(G)$ *and thus at least* c_n *nodes must be removed to disconnect* G*.*

Definition 3.2.4 *The* ***link-connectivity*** *of a graph* G *is the minimal number* c_l*, such that there are* c_l *link-disjoint paths between any pair of nodes and thus at least* c_l *links must be removed to disconnect* G*.*

For a Δ_G-regular graph G, the maximum value of both node and link-connectivity is Δ_G. Thereby a Δ_G-regular graph is **optimally connected** if $c_n = c_l = \Delta_G$.

Theorem 3.2.3 ***(Theorem 3.7 of [28])*** *Every node-transitive graph* G *has* $c_l = \Delta_G$*.*

Corollary 3.2.1 *Every CG has* $c_l = \Delta_\Gamma$*.*

Corollary 3.2.2 *Every symmetric CG is optimally connected.*

Graph connectivity determines the fault-tolerance in scenarios of random node failures.

Definition 3.2.5 *The* ***fault-tolerance*** *of a graph is the maximum number* t*, such that if any* t *nodes are removed, the resulting graph is still connected thus* $t = c_n - 1$ *for any graph except by the complete graph, where* $t = c_n$*. A* Δ_G*-regular graph* G *is called* ***optimally fault-tolerant*** *if* $t = \Delta_G - 1$*.*

Corollary 3.2.3 *Optimally connected graphs are also optimally fault-tolerant.*

Corollary 3.2.4 *Every symmetric CG is optimally fault-tolerant.*

3.2.3 *Moore bound*

An important issue in the design of network topologies is to connect the maximum number of nodes while keeping small length of paths and considering that the number of ports per node is limited. The well-known **Moore bound** states that for any Δ_G-regular graph the number of nodes, i.e. n, satisfies $n \leqslant \frac{\Delta_G(\Delta_G-1)^{D_G}-2}{\Delta_G-2}$ [10, Theorem 10.1]. Therefore

$$D_G \geqslant \frac{\log(n-1)}{\log(\Delta_G)}. \tag{2}$$

Graphs satisfying this lower-bound, for a given n and Δ_G, are called **Moore graphs**. In addition to low diameter, it is also important to achieve low average distance, i.e. D_{avg}, in order to keep low latency.

Theorem 3.2.4 ***(Theorem 5.2 of [7])*** *For any node-transitive graph, and thus for any CG, it has that*

$$\frac{nD_G}{2(n-1)} \leqslant D_{avg} \leqslant D_G. \tag{3}$$

3.2.4 *Load balancing*

This section presents an analysis of the impact of graph symmetry on load balancing. It is assumed a uniform traffic pattern and ideal routing. Thus, a unit of traffic is exchanged per time unit between any pair of nodes in the network; and the traffic is balanced equally across all the shortest paths.

The **link load** is the amount of traffic units traversing a link per time unit. Under uniform traffic, the amount of traffic units per time unit is equal to $\frac{n(n-1)D_{avg}}{2}$. The **average link load**, i.e. l_{avg}, is the ratio between the amount of traffic units to the graph size, i.e. m, then $l_{avg} = \frac{n(n-1)D_{avg}}{2m}$. For regular graphs $m = \frac{n\Delta_G}{2}$, then

$$l_{avg} = \frac{(n-1)D_{avg}}{\Delta_G}. \tag{4}$$

A network reaches the saturation point when some link has the maximum link load, i.e. l_{max}. The **average link utilization** at the saturation point, i.e. u_{avg}, measures the percentage of use of links and is given by

$$u_{avg} = \frac{l_{avg}}{l_{max}}. \tag{5}$$

A network is called **well-balanced** if $u_{avg} = 1$. Therefore, at the saturation point, links keep carrying the maximum link load. Assuming ideal routing, in link-transitive networks $l_{avg} = l_{max}$, then all links carry the same load and $u_{avg} = 1$.

Corollary 3.2.5 *Every symmetric CG is well-balanced.*

Assuming that end points can be connected to nodes in the network. The network may, for example, be a DCN, where the nodes are routers and the end points are servers. Then total number of ports per node is called **the node radix** and is given by $R = \Delta_G + \Delta_p$, where Δ_p denotes the number of end points that can be connected to each node. The maximum value of Δ_p without reaching the saturation point is determined by the link utilization [30], i.e.

$$\Delta_p \leqslant \frac{\Delta_G u_{avg}}{D_{avg}}. \tag{6}$$

For *well-balanced* networks $u_{avg} = 1$ and thus

$$\Delta_p \leqslant \frac{\Delta_G}{D_{avg}}. \tag{7}$$

3.3 PERFORMANCE AND ROBUSTNESS EVALUATION

3.3.1 *Families of Cayley graphs evaluated*

According to the topological properties presented in the previous section, this section presents a performance and robustness evaluation of the following families of CGs: *2-Dimensional (2D) Torus*, *Hypercube*, *Bubble-sort*, *Star*, *Transposition* and *Butterfly*. These families of CGs were selected for its wide application as topologies of communication networks [2, 4, 6, 27, 31, 32].

Recall from Theorem 2.2.1 that every group, and thus CG, can be represented as a subgroup of the symmetric group S_p, where p is a positive integer, see Definition 2.2.11. The evaluated families of CGs and its definition in terms of S_p are presented as follows:

- **2D Torus** ($T2(p)$). It is the CG of the subgroup of S_p, where p is even, with generating set $S = \{(1, 2, \ldots, \frac{p}{2}), (2, \frac{p}{2}, \ldots, 1), (\frac{p}{2}+1, \frac{p}{2}+2, \ldots, p), (\frac{p}{2}+2, p, \ldots, \frac{p}{2}+1)\}$.
- **Hypercube** ($H(p)$). It is the CG of the subgroup of S_p, where p is even, with generating set $S = \{(i, i+1) : 1 \leqslant i < p \text{ and } i \text{ is }\}$.
- **Bubble-sort** ($BS(p)$). It is the CG of S_p with generating set $S = \{(i, i+1) : 1 < i \leqslant p\}$.
- **Star** ($St(p)$). It is the CG of S_p with generating set $S = \{(1, i) : 1 < i \leqslant p\}$.
- **Transposition** ($Tr(p)$). It is the CG of S_p with generating set $S = \{(i, j) : 1 \leqslant i < p,\ 1 < j \leqslant p \text{ and } i < j\}$.
- **Butterfly** ($B(p)$). It is the CG of the subgroup of S_p with generating set $S = \{(1, 2, \ldots, p)^2, (1, 2, \ldots, p)^2(1, 2)\}$.

Note that CGs can be recursively constructed applying the set of permutations S to each node, beginning at the node representing the identity element, i.e. $(1, 2, \ldots, p)$.

The results presented in this section depends on the topological parameters presented in Table 1. The value of these parameters for the evaluated families of CGs are summarized in Table 2.

Table 2: Parameters of Cayley graphs used as network topologies.

Cayley graph family	Number of nodes (n)	Diameter (D_Γ)	$O(D_\Gamma)$	Degree (Δ_Γ)	$O(\Delta_\Gamma)$
2D Torus ($T2(p)$)	$\frac{p^2}{2}$	$2\left\lceil\frac{p}{2}\right\rceil$	$O(n^{1/2})$	4	$O(1)$
Hypercube ($H(p)$)	$2^{p/2}$	$\frac{p}{2}$	$O(\log(n))$	$\frac{p}{2}$	$O(\log(n))$
Bubble-sort ($BS(p)$)	$p!$	$\frac{p^2-p}{2}$	$O(\log(n)^2)$	$p-1$	$O(\log(n))$
Star ($St(p)$)	$p!$	$\left\lfloor\frac{p^2-p}{2}\right\rfloor$	$O(\log(n))$	$p-1$	$O(\log(n))$
Transposition ($Tr(p)$)	$p!$	$p-1$	$O(\log(n))$	$\frac{p^2-p}{2}$	$O(\log(n)^2)$
Butterfly ($B(p)$)	$2^{p/2}\left\lfloor\frac{p}{2}\right\rfloor$	$\left\lfloor\frac{3p}{4}\right\rfloor$	$O(1)$	4	$O(1)$

3.3.2 *Fault-tolerance and load balancing*

as it was explained in Section 3.2.1, every CG is node-transitive. Moreover, 2D Torus, Hypercube, Star and Transposition graphs are also link-transitive, i.e. symmetric, which provides optimal connectivity, fault-tolerance and load balancing, see Table 3.

From Section 3.2.4, in well-balanced networks, nodes keep carrying the maximum value of link load, i.e. l_{max}. Figure 6 shows values of the maximum link load supported by the evaluated families of CGs. These values were computed assuming uniform traffic and ideal routing.

Table 3: Symmetric properties of Cayley graphs

Cayley graph family	Connectivity	Fault-tolerance	Load balancing
2D Torus	Optimal node and link connectivity (symmetric)	Optimal	Well-balanced ($u_{avg} = 1$)
Hypercube			
Star			
Transposition			
Bubble-sort	Optimal link connectivity	Non-optimal	$u_{avg} \approx 0.8$
Butterfly			

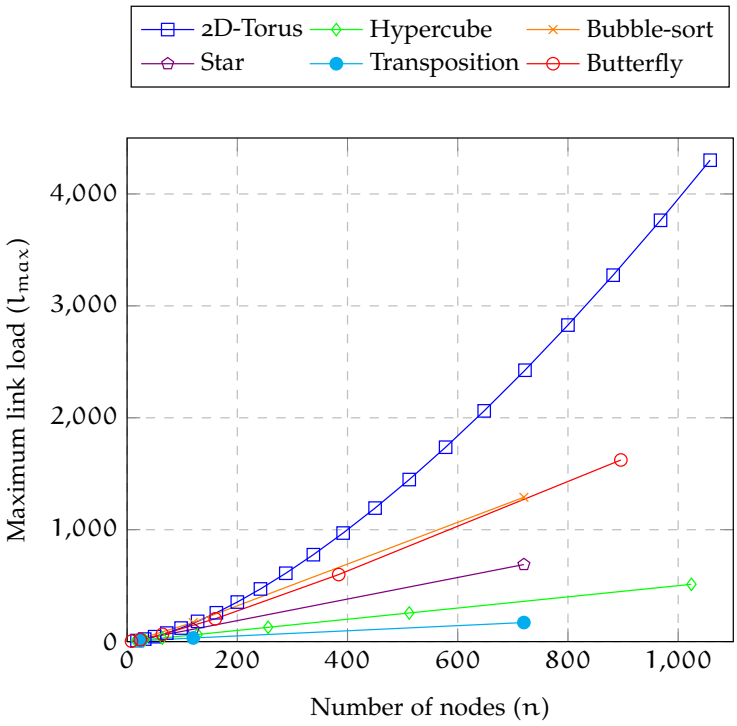


Figure 6: Maximum link load with respect to the number of nodes.

3.3.3 *Average distance vs. number of end points*

In the design of communication networks, it is important to find topologies close to the Moore bound (see Eq. (2) and Eq. (3)) in order to have low latency and support a high number of end points for a given node radix i.e. R. Figure 7 shows the average distance in the evaluated families of CGs with respect to their order. As it is shown, the family of Transposition graphs is the only one that satisfies the Moore bound, although the rest

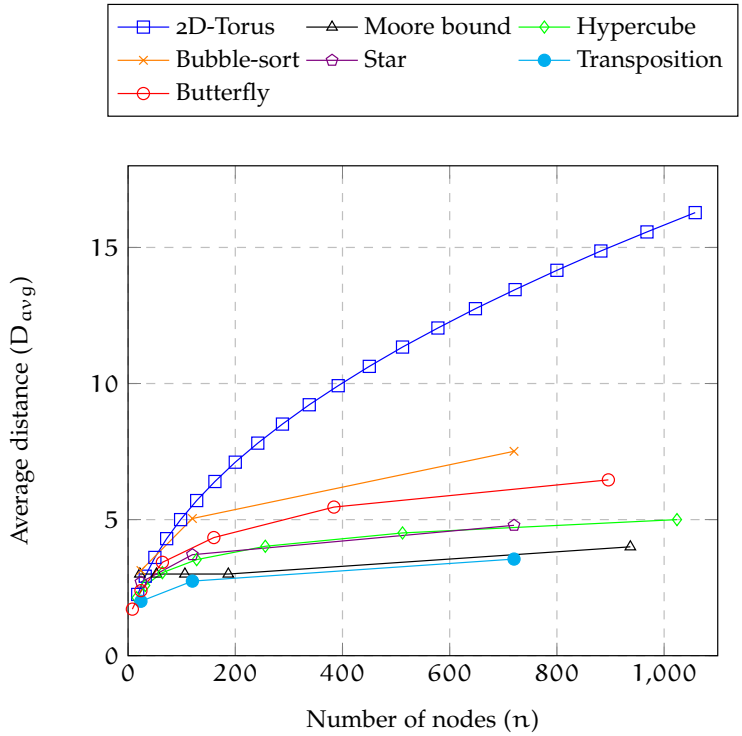


Figure 7: Average distance with respect to the number of nodes.

of CGs have logarithmic diameter except for 2D Torus graphs, which have the largest average distance and diameter (see Table 2). Large CGs that are close to the Moore bound can be found in [33, 34]

The Moore bound also determines the maximum number of nodes that can be connected for a given Δ_Γ. In addition, it is important to compute the number of end points that can be supported by the network, which is given by $p = n\Delta_p$. Figure 8 show the number of end points supported by each family of CGs with respect to their order. Comparing Figure 6 and Figure 8, the number of end points supported is inversely proportional to their maximum link load, which is congruent with Eq. (5) and Eq. (6). Transposition graphs are the topologies with lowest link load and therefore it support the maximum number of end points for a given number of nodes. Hypercube graphs supports $2n$ end point, then $\Delta_p = 2$. The remaining families of CGs supports the minimum number of end points, i.e. n, and the $\Delta_p = 1$.

Finally, Figure 9 shows the number of end points supported for a given node radix. Butterfly, Bubble sort and Star graphs support the largest number of end points with the lowest R. Actually, Butterfly graph has constant R. Except for Butterfly graphs, CGs have an exponential growth in their number of end points with respect to R. This feature poses an important challenge in the design of network topologies based on CGs, which consists in adding sequentially nodes without affecting the symmetric properties of CGs.

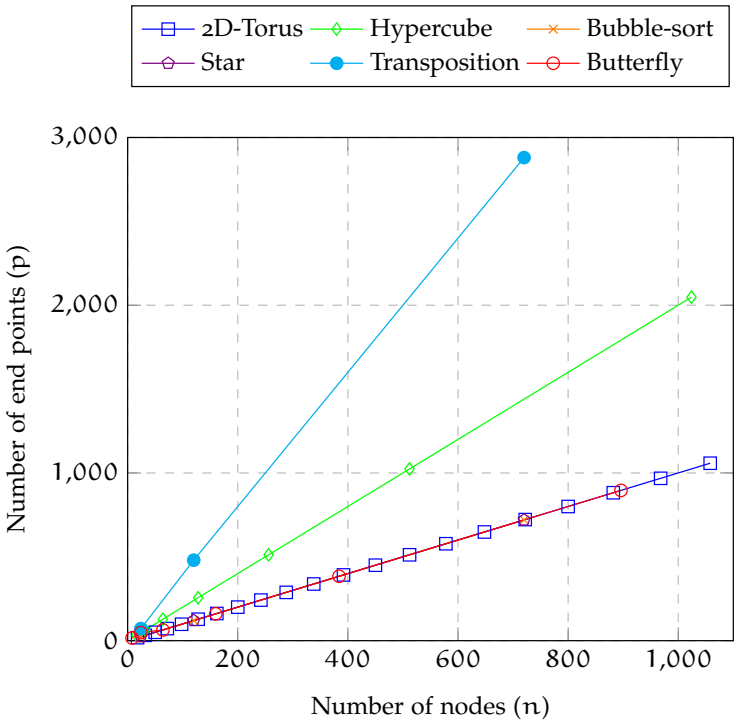


Figure 8: Number of end points supported with respect to the number of nodes.

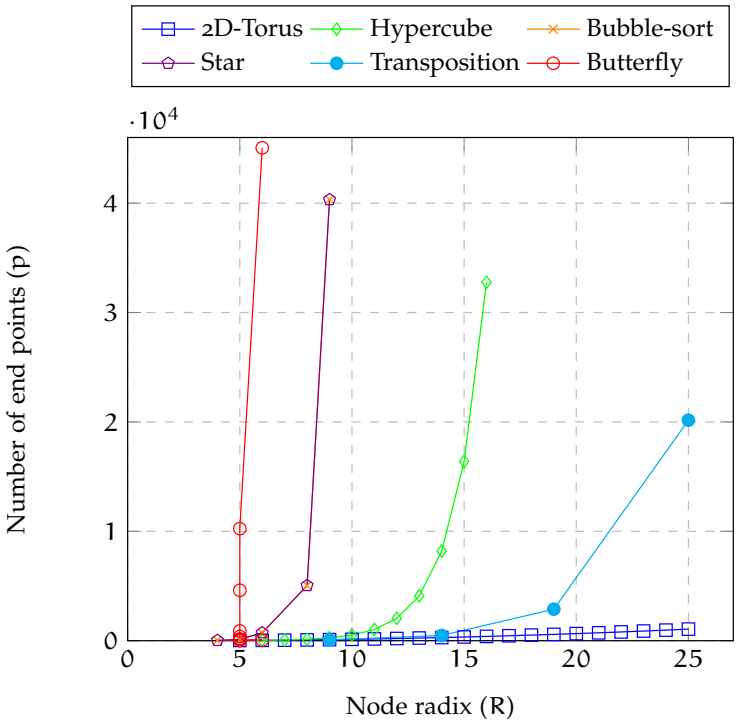


Figure 9: Number of end points supported with respect to the node radix.

Ordinary language is totally unsuited
for expressing what physics really asserts,
since the words of everyday life are not sufficiently abstract.
Only mathematics and mathematical logic
can say as little as the physicist means to say.

— Bertrand Russell

4

WORD PROCESSING IN CAYLEY GRAPHS

This Thesis employs word processing techniques to solve the routing problem in CGs. This approach arises from the AGT, whose fundamental idea states that CGs can be described by a linear recursion. Hence every CG can be constructed from repetitions of a finite subgraph of it. A formal definition of such recursion is given through a set of FSA that encode the topological structure of its related CG. From these FSA, low complexity algorithms can be constructed, e.g. path computation algorithms. This chapter explains the necessary concepts and terminology of the AGT for solving the shortest path problem in CGs. These concepts are the main theoretical base of this research work. For further details about word processing in CGs, refer to [25]

4.1 LANGUAGES AND AUTOMATA

This section presents some basic material from language and automata theory, which has connection with AGT. As a general reference on languages and automata theory, the reader can consult [35, 36]. Further material about language theory applied to algorithms on algebraic groups can be found in [24, 37].

4.1.1 *Words and languages*

Definition 4.1.1 *Let $\mathcal{A}$ be a finite set of symbols, hereafter $\mathcal{A}$ is referred to as an* ***alphabet*** *and its elements are referred to as* ***letters****. Any finite sequence of letters $x_1x_2\dots x_l$, such that $x_i \in \mathcal{A}$, is called a* ***word*** *or* ***string*** *over $\mathcal{A}$.*

Definition 4.1.2 *Let $w = x_1x_2\dots x_l$ be a word over $\mathcal{A}$, where the length of w, i.e. $|w|$, is equal to l. Then:*

- *a* ***substring of*** *w, denoted by $w(i)$, is given by the first i letters in w (if $i > |w|$, $w(i) = w$), and*
- *the* ***null string****, denoted by $e_{\mathcal{A}} \notin \mathcal{A}$, represents the word of length 0, i.e. $|e_{\mathcal{A}}| = 0$.*

Definition 4.1.3 *The set of all words over $\mathcal{A}$, including $e_{\mathcal{A}}$, is denoted by $\mathcal{A}^*$.*

Definition 4.1.4 *Any set of words $L \subseteq \mathcal{A}^*$ is called a* ***language*** *over $\mathcal{A}$.*

4.1.2 *Finite State Automata*

Definition 4.1.5 *A FSA, also called **automaton**, is a quintuple $(Q, \mathcal{A}, \delta, q_0, F)$, where Q is a finite set called the **state set**; $\mathcal{A}$ is an **alphabet**; $\delta : Q \times \mathcal{A} \to Q$ is a function called **transition function**; $q_0 \in Q$ is called the **initial state**; and F is a subset of Q called the **set of accept states**. Here and subsequently, $(Q, \mathcal{A}, \delta, q_0)$ denotes $(Q, \mathcal{A}, \delta, q_0, F)$ if $F = Q$; and q^x denotes $\delta(q, x)$, where $q \in Q$ and $x \in \mathcal{A}$.*

An automaton can be represented by a **state diagram**. Figure 10 presents an state diagram of the automaton $M_1 = (Q, \mathcal{A}, \delta, q_0, F)$, where:

1) $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$.

2) $\mathcal{A} = \{a, b, c\}$.

3) δ can be described by a state transition table as:

States \ Transitions	a	b	c
q_0	q_1	q_2	q_3
q_1	q_7	q_2	q_3
q_2	q_4	q_7	q_3
q_3	q_8	q_5	q_8
q_4	q_7	q_7	q_3
q_5	q_6	q_8	q_8
q_6	q_8	q_8	q_8
q_7	q_7	q_7	q_7
q_8	q_8	q_8	q_8

   An example of transition is $\delta(q_3, b) = q_5$ that is denoted by $q_3^b = q_5$.

4) The start state, i.e. q_0, is indicated by the arrow pointing at it from nowhere.

5) The accept states, i.e. $F = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$, are indicated by a double circle. Note that the states q_7 and q_8 are non accept states that go to themselves on every possible transition. These kind of states are called **dead states**.

An automaton can be described through its state diagram or state transition table due to these elements contain the same information of the automaton definition. An automaton with alphabet $\mathcal{A}$ is referred to as an **automaton over** $\mathcal{A}$, which can be seen as a machine that reads words over $\mathcal{A}$. The automaton starts in q_0 and reads a tape on which the word to be read is printed. The tape is read one letter at a time. After reading a letter, the state of the automaton is changed in accordance with the transition function, whose inputs are the existing state and the letter read. Then the tapehead is moved one letter to the right on the tape. If after reading all the string, the state of the automaton is in F, then the automaton answers *Yes*; Otherwise it answers *No*. For instance, when the automaton M_1, in Figure 10, reads the string $abac$, the processing proceeds as follows:

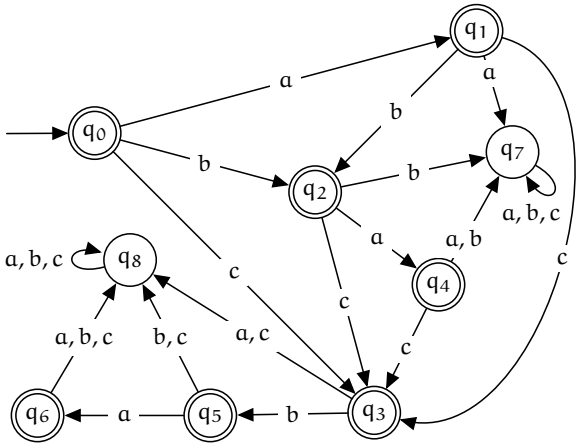


Figure 10: A finite state automaton called M_1. The language recognized by M_1 is $L(M_1) = \{e_{\mathcal{A}}, a, b, c, ab, ac, ba, bc, cb, aba, abc, acb, bac, bcb, cba, abac, abcb, acba, bacb, bcba, abacb, abcba, bacba, abacba\}$.

1) Start in state q_0.
2) Read a, follow transition from q_0 to q_1.
3) Read b, follow transition from q_1 to q_2.
4) Read a, follow transition from q_2 to q_4.
5) Read c, follow transition from q_4 to q_3.
6) *Yes* because M_1 is in an accept state q_3 after reading all the string.

Definition 4.1.6 *Let $M = (Q, \mathcal{A}, \delta, q_0, F)$ be a FSA. If after reading a word $w \in \mathcal{A}^*$, M answers Yes, it is said that w is **recognized** or **accepted** by M. Otherwise, it is said that w is **rejected** by M. The set of words recognized by M is called the **language recognized** by M, and is denoted by $L(M)$.*

Definition 4.1.7 *A language is called **regular language** if some FSA recognizes it.*

The language recognized by the automaton M_1 in Figure 10, i.e. $L(M_1)$, is a regular language. This language includes the *null string*, i.e. $e_{\mathcal{A}}$, due to the start state of M_1 is in an accept state.

4.1.3 *2-variable finite state automata*

The application of language theory in the study of algebraic groups involves automata that read simultaneously two words $u, v \in \mathcal{A}^*$, i.e. automata over $\mathcal{A} \times \mathcal{A}$. To dealt with words of unequal length, it is introduced an extra letter $\$ \notin \mathcal{A}$ called **padding symbol**, which is interpreted as the *null string*, i.e. $e_{\mathcal{A}}$.

Let $\mathcal{A}^+$ denote an alphabet given by $\mathcal{A} \cup \{\$\}$. Let $(u, v)^+$ denote a tuple $(x_1x_2 \dots x_l, y_1y_2 \dots y_l) \in (\mathcal{A}^+ \times \mathcal{A}^+)^*$, where $u, v \in \mathcal{A}^*$, $|u| = i$ and $|v| = j$ then:

(i) If $i = j$, $u = x_1x_2 \dots x_l$ and $v = y_1y_2 \dots y_l$. For example, if $u = abc$ and $v = cba$, then $(u, v)^+ = (abc, cba)$.

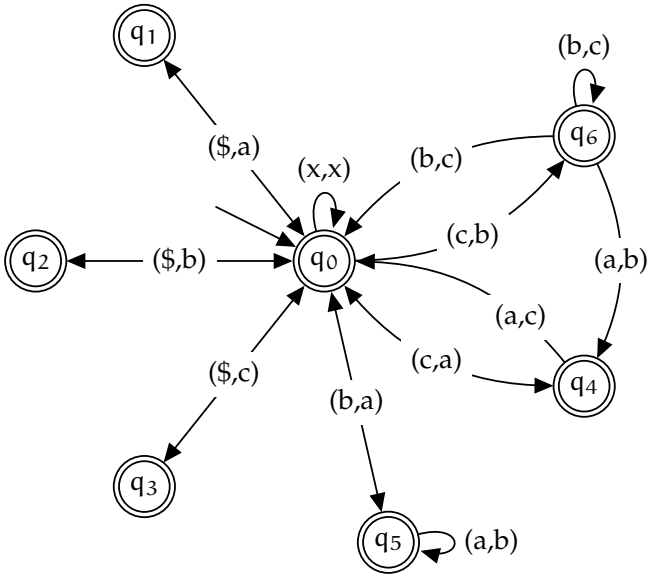


Figure 11: A 2-variable finite state automaton called M_2.

(ii) If $i < j$, $v = y_1 y_2 \dots y_l$ and $x_1 x_2 \dots x_l$ results from adding $l - i$ symbols \$ to u (in any position). For example, if $u = ab$ and $v = cba$, then $(u, v)^+ = (a\$b, cba) = (\$ab, cba) = (ab\$, cba)$.

(iii) If $i > j$, $u = x_1 x_2 \dots x_l$ and $y_1 y_2 \dots y_l$ results from adding $l - j$ symbols \$ to v (in any position). For example, if $u = abc$ and $v = c$, then $(u, v)^+ = (abc, \$\$c) = (abc, \$c\$) = (abc, c\$\$)$.

Definition 4.1.8 *A 2-variable FSA over $\mathcal{A}$ is an automaton with alphabet $\mathcal{A}^+ \times \mathcal{A}^+$, which recognizes tuples $(u, v)^+ \in (\mathcal{A}^+ \times \mathcal{A}^+)^*$ except by $(\$, \$)$.*

Figure 11 presents a 2-variable automaton called M_2. To simplify, only transitions to accept states are shown due to the rest of transitions lead to dead states. Examples of tuples recognized and rejected by M_2 are (ac, ca) and $(\$b, ca)$, respectively.

4.2 GROUPS AS LANGUAGES

Let $\mathcal{G} = \langle S|R \rangle$ be a finitely presented group and let $\mathcal{A}$ be an alphabet such that $|A| = |S \cup S^{-1}|$. Consider now a bijective map

$$\phi : \{S \cup S^{-1}\} \to \mathcal{A}, \tag{8}$$

which assigns each generator and its inverse to lowercase and uppercase variants of the same letter. A letter $X \in \mathcal{A}$ is said to be **the inverse of a letter** $x \in \mathcal{A}$ if and only if $X = \phi(s^{-1})$ and $x = \phi(s)$, where s^{-1} is the inverse of s.

Definition 4.2.1 *Let $w \in \mathcal{A}^*$, the following words are defined:*

- *The **inverse of** w, denoted by w^{-1}, is given by the reverse string of the inverse letters of w, e.g. if $w = aC$, then $w^{-1} = cA$.*

- *The **reduced form of** w, denoted by w_{red}, results from removing the substrings of the form uu^{-1} from w, e.g. if $w = baCcAa$ and $(aC)^{-1} = cA$, then $w_{red} = ba$.*

Definition 4.2.2 *Let $\mathcal{F}(\mathcal{A})$ be the **free group** over $\mathcal{A}$, which consists of the set of all reduced words over $\mathcal{A}$ (including $e_{\mathcal{A}}$). The group operation is the string concatenation where substrings of the form ww^{-1} are canceled. The identity element is $e_{\mathcal{A}}$.*

By Definition 2.2.6, there is an epimorphism given by the map

$$\pi : \mathcal{F}(\mathcal{A}) \to \mathcal{G}, \tag{9}$$

that assigns a unique set of words in $\mathcal{F}(\mathcal{A})$ to each group element in $\mathcal{G}$ [25, Section 2.1]. The symbol $e_{\mathcal{A}}$ is assigned to the identity element, i.e. ϵ.

Definition 4.2.3 *Let $=_{\mathcal{G}}$ denote an equivalence relation on $\mathcal{F}(\mathcal{A})$, such that $w =_{\mathcal{G}} v$ if and only if w and v represent the same group element under Eq. (9). The set of words representing the same group element is defined by the equivalence class $[w]$, i.e. $[w] = \{v \in \mathcal{F}(\mathcal{A}) : \pi(w) = \pi(v)\}$.*

In order to assign a unique word in $\mathcal{F}(\mathcal{A})$ to each element in $\mathcal{G}$ (and then to each node in $V(\Gamma)$), a canonical form of the set of equivalence classes $[w]$ needs to be defined.

Definition 4.2.4 *Let $<_{\mathcal{A}}$ be a lexicographical order over $\mathcal{A}$. Let $w, v \in \mathcal{A}^*$, it is said that w is ShortLex than v, if w is shorter than v, i.e. $|w| < |v|$, or w and v have the same length but w comes before v in the order $<_{\mathcal{A}}$.*

Definition 4.2.5 *The language of the ShortLex words in $\mathcal{F}(\mathcal{A})$ representing a unique group element in $\mathcal{G}$ is given by*

$$L = \{w \in \mathcal{F}(\mathcal{A}) : w <_{\mathcal{A}} v, \forall v \in \mathcal{F}(\mathcal{A}) \text{ s.t. } w =_{\mathcal{G}} v\}. \tag{10}$$

*This language is called the **language of the ShortLex words**.*

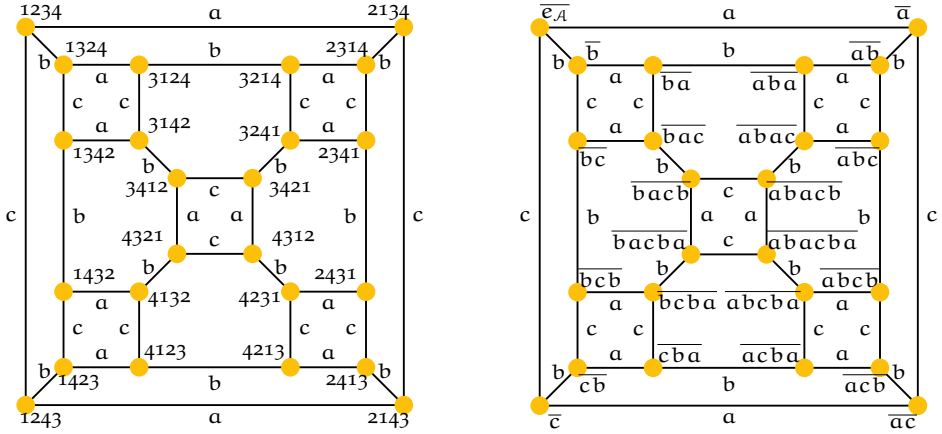
Therefore, there is a bijective map

$$\mu : \mathcal{G} \to L \tag{11}$$

that assigns each group element in $\mathcal{G}$ to its *ShortLex* representative word in $\mathcal{F}(\mathcal{A})$. Language L defines a canonical form for the set of equivalence classes of Definition 4.2.3.

4.3 WORDS AS PATHS AND NODES

From the CG definition (Definition 2.2.12), it is clear that a sequence of generators $s_0 s_1 \ldots$, where $s_i \in \{S \cup S^{-1}\}$, resulting in the group element $g \in \mathcal{G}$, represents a path in $\Gamma(\mathcal{G}, S)$ from the identity element, i.e. ϵ, to g. For example, consider the symmetric group S_4 with the generating set $S = \{(2134), (1324), (1243)\}$ see Definition 2.2.11. Figure 5 shows the CG of S_4, which is known as Bubble-sort graph and is denoted by $BS(4)$, see Section 3.3.1. The sequence $(2134)(1324)(1243)$ results in the permutation (2341), then it represents a path from (1234) to (2341), where (1234) is the identity permutation, i.e. $\epsilon = (1234)$.



(a) Edges labeled with letters representing its corresponding generator.

(b) Nodes labeled with words representing the *ShortLex* path from $\overline{e_{\mathcal{A}}}$ to each of them.

Figure 12: The Bubble-sort graph BS(4) with edges and nodes labeled.

Assume now that edges of a CG are labeled according to Eq. (8). Thus every path in the CG can be represented by a unique word $w \in \mathcal{F}(\mathcal{A})$, which defines a sequence of edges, where the symbol $e_{\mathcal{A}}$ represents the empty path. In particular if $w = \pi(g)$, then w represents a path from ϵ to w, meanwhile w^{-1} represents a path from w to ϵ. Returning to BS(4), let $\mathcal{A} = \{a, b, c\}$ be an alphabet and let ϕ denote the map:

$$\begin{aligned} \phi : \quad S \quad &\to \quad \mathcal{A} \\ (2134) \quad &\to \quad a \\ (1324) \quad &\to \quad b \\ (1243) \quad &\to \quad c \end{aligned} \tag{12}$$

Figure 12a shows BS(4) with their edges labeled according to Eq. (12). The word abc represents the sequence of edges $(2134)(1324)(1243)$ and thus a path from (1234) to (2341). Meanwhile, $(abc)^{-1} = cba$[1] represents the sequence of edges $(1243)(1324)(2134)$ and thus a path from (2341) to (1234). The set of words in $\mathcal{F}(\mathcal{A})$ representing all the paths between two nodes are defined as follows.

Lemma 4.3.1 *Words in the same equivalence class, i.e.* $[w]$*, represent all paths between the same pair of nodes.*

Proof. From Definition 4.2.3, words in an equivalence class $[w]$ represent the same group element under the group homomorphism defined by Eq. (9). Let $g \in \mathcal{G}$ be the element represented by $[w]$, then words in $[w]$ represent all paths from ϵ to g, where ϵ is the identity element (see Definition 2.2.1). Since CGs are node-transitive, words $[w]$ represent all paths between every pair of nodes $g_1, g_2 \in V(\Gamma)$, such that $g_2 = g \cdot g_1$. ∎

In Figure 12a, the words abc and $babca$ are in the same equivalence class as they represent the same permutation and thus the same group element (2341). From Lemma 4.3.1, both words represent paths from (1234)

1 In the case of S_4, $S = S^{-1}$ and then the inverse each letter in $\mathcal{A}$ is itself, i.e. $A = a$, $B = b$ and $C = c$.

to (2341), from (2341) to (3412), and in general, from nodes g_1 to g_2 such that $g_2 = (2341)g_1$.

Consider now a word $w \in \mathcal{F}(\mathcal{A})$ representing a group element $g \in \mathcal{G}$, i.e. $\pi(w) = g$ according to Eq. (9), and then a path from ϵ to g in $\Gamma(\mathcal{G}, S)$. Hereafter $\widehat{w}$ denotes a path represented by the word w, and $\overline{w}$ denotes the node g, where $e_{\mathcal{A}}$ is assigned to ϵ. Figure 12b shows BS(4) with each node g labeled as $\overline{w}$, where w is the shortest word, such that $\pi(w) = g$. Note that $\widehat{w}$ can represent a path beginning in any node. In order to refer a specific path in this notation, the first node in the path must be specified. Hereafter $\widehat{w}(\overline{w_1})$ denotes a path representing by w and beginning at node $\overline{w_1}$, where $\widehat{w}(\overline{w_1})$ is given by a list of nodes $[\overline{w_1}, \ldots, \overline{w_l}]$. In particular, $\widehat{w}(\overline{e_{\mathcal{A}}})$ represents a path from $\overline{e_{\mathcal{A}}}$ to $\overline{w}$. In Figure 12b, the path $\widehat{abc}(\overline{b})$ is given by the list of nodes $[\overline{b}, \overline{ba}, \overline{aba}, \overline{abac}]$, meanwhile the path $\widehat{abc}(\overline{e_{\mathcal{A}}})$ is given by $[\overline{e_{\mathcal{A}}}, \overline{a}, \overline{ab}, \overline{abc}]$.

From the above explanation, there is a path between any two nodes $\overline{u}$ and $\overline{v}$, denoted by the word $u^{-1}v$. The path $\widehat{u^{-1}v}(\overline{u})$ goes from $\overline{u}$ to $\overline{e_{\mathcal{A}}}$ and from $\overline{e_{\mathcal{A}}}$ to $\overline{v}$. In Figure 12b, a path from $\overline{cb}$ to $\overline{acb}$ is given by $\widehat{bcacb}(\overline{cb})$ where $bcacb = (cb)^{-1}acb$. Note that the problem of computing the shortest path between $\overline{u}$ and $\overline{v}$ is equivalent to computing the shortest word $w \in \mathcal{F}(\mathcal{A})$, such $w =_{\mathcal{G}} u^{-1}v$, which is the canonical form of $u^{-1}v$. Recall from the previous section that the canonical form of any word in $\mathcal{F}(\mathcal{A})$ is in the language of the *ShortLex* words in $\mathcal{F}(\mathcal{A})$, see Definition 4.2.5. Then:

Corollary 4.3.1 *The language of the ShortLex words, i.e.* L, *related to a group* $\mathcal{G} = (S|R)$, *gives a unique representation for the shortest paths between any pair of nodes in* $\Gamma(\mathcal{G}, S)$. *From now on, paths represented by words in* L *are called ShortLex paths.*

Continuing with BS(4), let $a < b < c$ be a lexicographic order over the alphabet $\mathcal{A} = \{a, b, c\}$. From Definition 4.2.5, the language of the *ShortLex* words of BS(4) is

$$\begin{aligned} L = \{&e_{\mathcal{A}}, a, b, c, ab, ac, ba, bc, cb, aba, abc, acb, \\ &bac, bcb, cba, abac, abcb, acba, bacb, \\ &bcba, abacb, abcba, bacba, abacba\} \end{aligned} \tag{13}$$

From Corollary 4.3.1, any of the shortest paths in BS(4) (Figure 12b) can be represented by a word in Eq. (13). As it was mentioned above, a path from $\overline{cb}$ to $\overline{acb}$ is given by $\widehat{bcacb}(\overline{cb})$, where $\pi(bcacb) = (3214)$. However the *ShortLex* path from $\overline{cb}$ to $\overline{acb}$ is $\widehat{aba}(\overline{cb})$, where $aba \in L$ is the canonical from of $bcacb$ as aba is the *ShortLex* word representing (3214). The problem of reducing any word $u^{-1}v \in \mathcal{F}(\mathcal{A})$ to its canonical form is called the Minimum Word Problem (MWP) and can be solved in time $O(|u^{-1}v|^2)$ for *ShortLex* Automatic Groups (SAGs) [25, Theorem 2.3.10].

4.4 SHORTLEX AUTOMATIC GROUPS

Definition 4.4.1 (*ShortLex* automatic group) *Let* $\mathcal{G} = \langle S|R \rangle$ *be a finitely presented group, then* $\mathcal{G}$ *is a* SAG *if it admits a* SAS, *which consists of:*

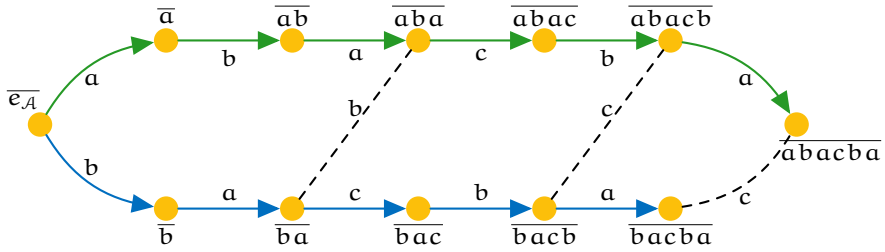


Figure 13: Geometric representation of paths $\widehat{abacba}(\overline{e_{\mathcal{A}}})$ (green) and $\widehat{bacba}(\overline{e_{\mathcal{A}}})$ (blue) in $BS(4)$. Both paths follow the k-*fellow-traveler* property as their uniform distance is 2.

(i) A lexicographically ordered alphabet $\mathcal{A}$, such that S and $\mathcal{A}$ satisfy Eq. (8).

(ii) A FSA called Word-Acceptor (WA) and denoted by W, such that $L(W)$ is the language of the ShortLex words defined by Eq. (10).

(iii) A 2-variable FSA for each $x \in \mathcal{A} \cup \{e_{\mathcal{A}}\}$ called Multiplier Automata (MA) and denoted by M_x, such that $(r,t)^+ \in L(W) \times L(W)$ is accepted by M_x if and only if $rx =_{\mathcal{G}} t$.

Henceforth, the SAS of $\Gamma(\mathcal{G},S)$ will be denoted by $(\mathcal{A},L)$.

Examples of SAGs are the groups presented in Section 3.3.1. Taking the CG in Figure 12b as example, its WA is presented in Figure 10. An important feature of SAGs is that their associated CGs follow the k-*fellow-traveler* property.

Lemma 4.4.1 (k-fellow-traveler property) *Let $\mathcal{G} = \langle S|R\rangle$ be a SAG and let $\widehat{r}$ and $\widehat{t}$ be two ShortLex paths in $\Gamma(\mathcal{G},S)$ beginning at the same node and whose end nodes are adjacent. The **uniform distance** between $\widehat{r}$ and $\widehat{t}$, denoted by $d_\Gamma(\widehat{r},\widehat{t})$, is given the maximum value of $d_\Gamma(\overline{r(i)},\overline{t(i)})$, see Definition 2.1.9, for all nodes $\overline{r(i)}$ and $\overline{t(i)}$ in $\widehat{r}$ and $\widehat{t}$, respectively. Then, it is said that $\Gamma(\mathcal{G},S)$ satisfies the k-fellow-traveler property, if there exists a constant k (depending on $\mathcal{G} = \langle S|R\rangle$), such that the uniform distance between all ShortLex paths $\widehat{r}$ and $\widehat{t}$ is bounded by k, i.e. $d_\Gamma(\widehat{r},\widehat{t}) \leqslant k$.*

Note that distances between paths are bounded in any finite CGs. Therefore, any finite group is a SAG. In the case of infinite groups, only groups having the k-*fellow-traveler* property are SAGs. To illustrate the k-*fellow-traveler* property, consider the paths $\widehat{abacba}(\overline{e_{\mathcal{A}}})$ and $\widehat{bacba}(\overline{e_{\mathcal{A}}})$ in $BS(4)$, whose end nodes, i.e. $\overline{abacba}$ and $\overline{bacba}$, are adjacent. Both paths are *ShortLex* as they are represented by words in L, see Eq. (13). Figure 13 shows that the uniform distance between these paths is bounded by 2, hence they follow the k-*fellow-traveler* property.

Definition 4.4.2 *Let $\Gamma(\mathcal{G},S)$ be a CG with fellow-traveler constant k and SAS $(\mathcal{A},L)$. Then, the **set of Word-Differences (WDs)** of $\Gamma(\mathcal{G},S)$ is given by the set of ShortLex words whose length is less or equal than k, i.e.*

$$\mathbb{D} = \{w \in L : |w| \leqslant k\}. \tag{14}$$

Note that the words in $\mathbb{D}$ are the labels of nodes in the k-neighbourhood of node $\overline{e_{\mathcal{A}}}$, i.e. $N(\overline{e_{\mathcal{A}}},k)$, see Definition 2.1.11. In the case of $BS(4)$, $k = 2$

and then $\mathbb{D} = \{e_{\mathcal{A}}, a, b, c, ab, ac, ba, bc, cb\}$. From Lemma 4.4.1 and Definition 4.4.2, for every pair of paths $\widehat{r}$ and $\widehat{t}$ that have the k-*fellow-traveler* property, the *ShortLex* path between any pair of nodes $\overline{r}(i)$ and $\overline{t}(i)$ is given by a word in $\mathbb{D}$. As it was mentioned above, paths $\widehat{abacba}(\overline{e_{\mathcal{A}}})$ and $\widehat{bacba}(\overline{e_{\mathcal{A}}})$ in BS(4) follow the k-*fellow-traveler* property. From Figure 13, the *ShortLex* paths from $\overline{abacba}(i)$ to $\overline{bacba}(i)$, for $1 \leqslant i \leqslant 6$, are: $\widehat{ab}(\overline{a})$, $\widehat{ab}(\overline{ab})$, $\widehat{bc}(\overline{aba})$, $\widehat{bc}(\overline{abac})$, $\widehat{ac}(\overline{abacb})$, and $\widehat{c}(\overline{abacba})$, respectively. All of these paths are represented by words in $\mathbb{D}$.

A 2-variable FSA that recognizes words whose associated paths follow the k-*fellow-traveler* property is defined as follows.

Definition 4.4.3 *Let $\mathcal{G} = \langle S|R\rangle$ be a SAG with SAS $(\mathcal{A}, L)$. The Word-Difference Automaton (WDA) of $\mathcal{G} = \langle S|R\rangle$ with respect to $(\mathcal{A}, L)$ is the 2-variable FSA over $\mathcal{A}$ (Definition 4.1.8) consisting of:*

(i) An alphabet $\mathcal{B} = \mathcal{A}^+ \times \mathcal{A}^+$, where $\mathcal{A}^+ = \mathcal{A} \cup \{\$\}$ and the padded symbol $\$$ is interpreted as $e_{\mathcal{A}}$.

(ii) A set of states given by $\mathbb{D} \cup \{q_d\}$, where $e_{\mathcal{A}}$ is the start state and q_d denotes a dead state. The set of accept states is $\mathbb{D}$.

(iii) A transition function $\delta : \mathbb{D} \times \mathcal{B} \to \mathbb{D} \cup \{q_d\}$, such that $q_i^{(x,y)} = q_j$ for $q_j \in \mathbb{D}$ if and only if $q_j =_{\mathcal{G}} Xq_iy$, where $X = x^{-1}$.

This automaton is denoted by Diff $= (\mathbb{D}, \mathcal{B}, \delta, e_{\mathcal{A}})$. The automaton accepts tuples $(r, t)^+$, where $r, t \in \mathcal{A}^*$, if and only if the canonical form of any word $r(i)^{-1}t(i)$ is in $\mathbb{D}$. The state after reading $(r(i), t(i))^+$, denoted by $q^{(r(i),t(i))}$, is given by the canonical form of $r(i)^{-1}t(i)$, i.e. $q^{(r(i),t(i))} =_{\mathcal{G}} r(i)^{-1}t(i)$.

From a geometric point of view, if (r, t) is accepted by *Diff*, then $q^{\widehat{(r(i),t(i))}}$ is the *ShortLex* path between end nodes of paths $\widehat{r(i)}$ and $\widehat{t(i)}$ beginning at the same node. Table 4 presents the WDA of BS(4). When this automaton reads the tuple $(abacba, bacba\$)$, each transition result in a state representing a path as follows:

1. Start in state $e_{\mathcal{A}}$. Then words to be read represent path beginning in the same node.
2. Read (a, b), follow transition from $e_{\mathcal{A}}$ to ab. Then $\widehat{ab}$ is the *ShortLex* path between the end nodes of $\widehat{a}$ and $\widehat{b}$.
3. Read (b, a), follow transition from ab to ab. Then $\widehat{ab}$ is the *ShortLex* path between the end nodes of $\widehat{ab}$ and $\widehat{ba}$.
4. Read (a, c), follow transition from ab to bc. Then $\widehat{bc}$ is the *ShortLex* path between the end nodes of $\widehat{aba}$ and $\widehat{bac}$.
5. Read (c, b), follow transition from bc to bc. Then $\widehat{bc}$ is the *ShortLex* path between the end nodes of $\widehat{abac}$ and $\widehat{bacb}$.
6. Read (b, a), follow transition from bc to ac. Then $\widehat{ac}$ is the *ShortLex* path between the end nodes of $\widehat{abacb}$ and $\widehat{bacba}$.
7. Read $(a, \$)$, follow transition from ab to c. Then $\widehat{c}$ is the *ShortLex* path between the end nodes of $\widehat{abacba}$ and $\widehat{bacba}$.

Table 4: State transition table for the word-difference automaton of $\mathrm{BS}(4)$. The start state is $e_{\mathcal{A}}$. All states are accepted except for the dead state q_d.

States	Transitions													
	(a,a)	(a,b)	(a,c)	$(\$,a)$	(b,a)	(b,b)	(b,c)	$(\$,b)$	(c,a)	(c,b)	(c,c)	$(\$,c)$	$(a,\$)$	$(b,\$)$	$(c,\$)$
$e_{\mathcal{A}}$	$e_{\mathcal{A}}$	ab	ac	a	ba	$e_{\mathcal{A}}$	bc	b	ac	cb	$e_{\mathcal{A}}$	c	a	b	c
a	a	b	c	$e_{\mathcal{A}}$	b	q_d	q_d	ba	c	q_d	a	ac	$e_{\mathcal{A}}$	ab	ac
b	q_d	a	q_d	ab	a	b	c	a	q_d	c	q_d	cb	ba	$e_{\mathcal{A}}$	bc
c	c	q_d	a	ac	q_d	q_d	b	bc	a	b	c	a	ac	cb	a
ba	ab	ba	q_d	q_d	$e_{\mathcal{A}}$	ab	ac	a	cb	q_d	q_d	q_d	b	q_d	q_d
ac	ac	cb	$e_{\mathcal{A}}$	c	bc	q_d	ba	q_d	$e_{\mathcal{A}}$	ab	ac	a	c	q_d	a
cb	q_d	ac	q_d	q_d	q_d	bc	cb	q_d	ba	$e_{\mathcal{A}}$	bc	b	q_d	c	q_d
ab	ba	$e_{\mathcal{A}}$	bc	b	ab	ba	q_d	q_d	q_d	ac	q_d	q_d	q_d	a	q_d
bc	q_d	q_d	ab	q_d	ac	cb	$e_{\mathcal{A}}$	c	q_d	bc	cb	q_d	q_d	q_d	b

These paths are illustrated in Figure 13 for $\widehat{abacba}$ and $\widehat{bacba}$ beginning at $\overline{e_{\mathcal{A}}}$[2]. The WDA is of major importance in the solution of the MWP, thereby it is necessary to compute it. Given a group presentation of a SAG, i.e. $\mathcal{G} = \langle S|R\rangle$, and a lexicographically ordered alphabet $\mathcal{A}$ satisfying Eq. (8), the Knuth-Bendix completion algorithm [38] is able to compute the SAS of $\mathcal{G} = \langle S|R\rangle$ and the WDA. The software package Knuth-Bendix on Monoids and Automatic Groups (KBMAG) [39] implements the Knuth-Bendix completion algorithm, details of its time requirements can be found in [37, Section 13.3.6].

A note of caution is required here as for some SAGs it is not possible to compute its SAS for any lexicographic order [37, Section 13.2.1]. For instance Coxeter groups, such as the group related to the Bubble-sort graph, are *ShortLex* automatic only with some lexicographic orders. On the other hand, SASs of *word hyperbolic* groups [40] can be computed for any lexicographic order.

4.5 SOLVING THE MINIMUM WORD PROBLEM

The problem of reducing a word to its canonical form is known as the MWP. The computation of the canonical form of a word consists in applying a set of rewriting rules to it. These rules have the form $r \to t$, where $r, t \in \mathcal{A}^*$, $r =_{\mathcal{G}} t$ and $r <_{\mathcal{A}} t$. From Definition 4.4.3, the rewriting rules of a SAG are encoded in its WDA.

Given the automaton *Diff* and a word $z \in \mathcal{A}^*$, Algorithm 1 presents the steps to reduce z to its canonical form $w \in L$. First, w is initialized to z_{red}, see Definition 4.2.1, (line 1). Then *while* loop (at line 2) is executed while w is not in canonical form, i.e. when w has a substring $w(i)$ that is not in canonical form. Recall that if $w(i)$ is not in canonical form, then it exists $r \in L$, such that $q^{(r,w(i))^+} = e_{\mathcal{A}}$. The word r can be search by exploring transitions of *Diff*, see [37, Section 13.1.7]. In this case, $w(i)$ is replaced by its canonical form r in w (*if/else* conditional at line 3). The resulting w is reduced (line 7) and the loop is repeated until every $w(i)$ is in canonical form and thus w is also in canonical form.

Example 4.5.1 *Consider the word* $z = abcababacba$ *and the WDA defined by the state transition table presented in Table 4. To reduce* z *to its canonical form* $w \in L$, *Algorithm 1 proceeds as follows:*

1) $w \leftarrow abcababacba$.
2) $w(4) = abca$ *is not in canonical form and* $q^{(r,w(4))} = e_{\mathcal{A}}$ *for* $r = abac \in L$, *then:*
   a) *Replace* $w(4)$ *by* r *in* $\underline{abca}babacba$, *i.e.* $w \leftarrow \underline{abac}babacba$.
   b) w *is in reduced form.*
3) $w(7) = abacbab$ *is not in canonical form and* $q^{(r\$\$,w(7))} = e_{\mathcal{A}}$ *for* $r = abcba \in L$, *then:*
   a) *Replace* $w(7)$ *by* r *in* $\underline{abacbab}acba$, *i.e.* $w \leftarrow \underline{abcba}acba$.

2 The resulting paths are valid for $\widehat{abacba}$ and $\widehat{bacba}$ beginning in any other node.

b) w is not in reduced form, then remove aa from w, i.e. $w \leftarrow abcbcba$.

4) $w(5) = abcbc$ is not in canonical form and $q^{(r\$\$,w(5))} = e_{\mathcal{A}}$ for $r = acb \in L$, then:

a) Replace $w(5)$ by r in $\underline{abcbc}ba$, i.e. $w \leftarrow \underline{acb}ba$.

b) w is not in reduced form, then remove bb from w, i.e. $w \leftarrow aca$.

5) $w(3) = aca$ is not in canonical form and $q^{(r\$\$,w(3))} = e_{\mathcal{A}}$ for $r = c \in L$, then:

a) Replace $w(3)$ by r in $\underline{aca}$, i.e. $w \leftarrow c$.

b) w is in reduced form.

6) $w = c$ is in canonical form.

7) Return w.

Algorithm 1 Compute the canonical form of a word.

Input: The automaton $\mathit{Diff} = (\mathbb{D}, \mathcal{B}, \delta, e_{\mathcal{A}})$.
Input: A word $z \in \mathcal{A}^*$.
Output: The word $w \in L$ such that $w =_G z$.

1: $w \leftarrow z_{red}$.
2: **while** w has a substring $w(i)$ such that $q^{(r,w(i))} = e_{\mathcal{A}}$ and $r <_{\mathcal{A}} w(i)$ for some $r \in L$ **do**
3: **if** $r = e_{\mathcal{A}}$ **then**
4: Remove $w(i)$ from w.
5: **else**
6: Replace $w(i)$ with r in w.
7: $w \leftarrow w_{red}$.
8: **return** w.

Lemma 4.5.1 *Algorithm 1 computes the canonical form of a word $z \in \mathcal{A}^*$ in time $O(|\mathbb{D}|\Delta_\Gamma|z|^2)$.*

Proof. The *while* loop is executed for each substring $w(i)$ that is not in canonical form, i.e. $O(|z|)$ times. For each $w(i)$, the search of a word $r \in L$ such that $q^{(r,w(i))} = e_{\mathcal{A}}$ can be done by traversing each state in $\mathbb{D}$ at most $|w(i)| \leqslant |z|$, i.e. $O(|\mathbb{D}|\Delta_\Gamma|z|)$ times [37, Section 13.1.7]. Therefore, Algorithm 1 computes the canonical form of z in time $O(|\mathbb{D}|\Delta_\Gamma|z|^2|)$. ■

5 STATE OF THE ART ON ROUTING IN CAYLEY GRAPHS

This Thesis proposes a generic routing scheme for CGs, i.e. a scheme that works on several families of CGs. Thereby this chapter summarizes and compares the main proposals of generic routing for CGs, which are classified into path computation algorithms and routing schemes, see Section 1.2.2.1. It is important to mention that there are several routing proposals specialized in a family of CGs, such that the routing scheme for the family of Pancake graphs [41].

The path computation algorithms analyzed are the Sims Factoring Algorithm (SFA) [17], which works on any CG; and the Path Computation Algorithm for Abelian Cayley Graphs (PCAACG) [18]. The routing schemes analyzed are the Routing based on Permutation Sort (RPS) [1], the Routing based on Chordal Ring Representation (RCRR) [16] and the Geometric Routing with Word-Metric Spaces (GRWMS) [15], which is extended and enhanced in this Thesis. These proposals are described following the topology model and notation presented in Section 3.1. Their performance have been evaluated on the families of CGs introduced in Table 2 according to the complexity metrics presented in Section 1.2.3.

5.1 PATH COMPUTATION ALGORITHMS

5.1.1 *Sims factoring algorithm*

The SFA computes not only the shortest path in CGs, but also the minimal paths (see Section 2.1.2). The algorithm applies concepts of word processing in CGs presented in Section 4.2 and Section 4.3. Thereby nodes and paths are represented by words in L and to compute the shortest path between any pair of nodes $u, v \in L$ is equivalent to factorize, i.e. to reduce, the word $u^{-1}v$ to a word w with minimal length, such that $w =_G u^{-1}v$. For reducing words, the algorithm takes as base the Schreier–Sims algorithm [42], where the factorization rules are encoded in arrays called Schreier vectors. The size of a Schreier vector is $O(\Delta_\Gamma \log(\Delta_\Gamma))$ bits. The shortest path is computed using $O(\Delta_\Gamma^2)$ vectors, whereas the minimal paths are computed using $O(\Delta_\Gamma^3)$ vectors. In both cases, the algorithm runs in time $O(\Delta_\Gamma^5)$.

5.1.2 *Path computation algorithm for abelian Cayley graphs*

The PCAACG computes the node-disjoint paths from a source node to $K \leqslant \Delta_\Gamma$ (not necessarily different) destination nodes in CGs of abelian groups, i.e. abelian CGs. The strategy followed is to map the destination nodes in the abelian CG to nodes in the Δ_Γ-dimensional hypercube and then to compute the node-disjoint paths from the identity node to each node in the Δ_Γ-dimensional hypercube. Then the searched paths are de-

rived from the paths computed in the Δ_Γ-dimensional hypercube. The PCAACG consists of three algorithms for: 1) proving the existence of the node-disjoint paths to be computed, which takes $O(K\Delta_\Gamma^{3/2})$ time units; 2) computing the shortest paths in hypercubes, which takes $O(K\Delta_\Gamma^{3/2})$, and 3) deriving paths in abelian CGs from paths in hypercubes, which takes $O(K\Delta_\Gamma)$ time units. The space complexity of these algorithms, and thus of the PCAACG is $O(K\Delta_\Gamma)$.

5.1.3 *Comparison of path computation algorithms for Cayley graphs*

This section presents a comparison of the algorithms previously presented. The main difference between them is that the SFA works on any CG; meanwhile, the PCAACG is limited to families of abelian CGs. Regarding the computed paths, both algorithms compute the shortest path. In addition, the PCAACG is able to compute the node-disjoint paths from a source node to K destination nodes, which can be the same or different. On the other hand, the SFA is able to compute just the minimal paths between two nodes. Note that if the node-disjoint paths are not minimal (see Section 2.1.2), then the SFA is not able to compute them. Regarding to the time complexity, the PCAACG is faster than SFA and also has lower space complexity. Table 5 summarizes the main features of these algorithms. As it is shown both space and time complexities depends just on the node degree.

Table 5: Summary of path computation algorithms for Cayley graphs.

Algorithm	Group family	Computed paths	Space complexity	Time complexity
SFA	Any group	The shortest path	$O(\Delta_\Gamma^3 \log(\Delta_\Gamma))$	$O(\Delta_\Gamma^5)$
		The minimal paths	$O(\Delta_\Gamma^4 \log(\Delta_\Gamma))$	
PCAACG	Abelian groups	The shortest path	$O(\Delta_\Gamma)$	$O(\Delta_\Gamma^{3/2})$
		The node-disjoint paths	$O(\Delta_\Gamma^2)$	$O(\Delta_\Gamma^{5/2})$

Table 6 and Table 7 show the space and time complexities, respectively, of the SFA and the PCAACG running on the CGs presented in Table 2. These results are given in terms of the number of nodes, i.e. n, substituting the values of Δ_Γ presented in Table 2 into the expressions of the time and complexity presented in Table 5. The complexities of the PCAACG were computed just for 2D Torus and Hypercube graphs as these CGs are the only abelian among the evaluated CGs.

As it is expected both algorithms have constant space and time complexities when running on CGs with constant degree, such as 2D Torus and Butterfly graphs. Conversely, the SFA has the highest space and time complexities in Transposition graphs, which also have the highest value of node degree among the evaluated CGs, i.e. $O(\log(n)^2)$. Finally, the space

and time complexities are the same when the algorithms runs in Hypercube, Bubble-sort and Star graphs due to the value of the node degree in these graphs grows at the same ratio, i.e. $O(\log(n))$.

5.2 ROUTING SCHEMES

5.2.1 *Routing based on permutation sort*

The RPS is supported on the Cayley's theorem, which states that every group is isomorphic to a subgroup of the symmetric group (Theorem 2.2.1). From Definition 2.2.11, the symmetric group, i.e. S_p, consists of a set of permutations of an array of p integers and the group operation is the composition of permutations. In this scheme, nodes are labeled with arrays (representing permutations), hence a node label can be represented by $O(p\log(p))$ bits. The connection rules between nodes are given by the set of permutations representing the generating set, thereby the shortest path problem being equivalent to finding an optimal set of permutations that lead from a source node to a destination node [1].

Usually this scheme follows a greedy routing strategy, where message headers include the label of the destination node, and the routing table of a node consists of the labels of its adjacent nodes. To forward packets, nodes compute the distance (in terms of the number of permutations) from each of its adjacent nodes to the destination node. Then, packets are forwarded to the nearest node to the destination node. Assuming that comparing two node labels takes a total amount of time proportional to the label size, the forwarding decision takes $O(\Delta_\Gamma p\log(p))$ time units. As examples of RPS, it can mention the schemes designed to work on Pancake, Star [1], Hypercube and Butterfly graphs [22], respectively.

5.2.2 *Routing based on chordal ring representations*

Authors in [16] prove that CGs of Borel subgroups, also called Borel CGs [5, 23, 43, 44], can be represented by Generalized Chordal Rings (GCRs). In a GCR, nodes can be labeled with integers from 0 to $n-1$, and there is a divisor q of n such that every two nodes i and j are connected if node $i+q$ module n is connected to node $j+q$ module n. In the RCRR, nodes of a Borel CG are mapped to nodes of a GCR. Then static routing tables are constructed following the connection rules of the GCR. The routing table of a node i consists of an entry of Δ_Γ bits for each destination node, where the j-th entry indicates the links that lead to the minimal paths from i to j. Routing is achieved identifying outgoing links for any incoming message, whose header consists of the destination label.

This routing scheme was extended in [45] to support link failures. In the new version, the j-th entry in the static routing table (of a node i) indicates the distance between j and each node adjacent to i. In addition, nodes incorporate a dynamic routing table, which indicates the distances to destination nodes taking into account link failures. The routing process consists of two stages, the first one proceeds as explained above. The second stage begins when a node can no longer forward a packet due to

Table 6: Space complexity of path computation algorithms on specific families of Cayley graphs.

Cayley graph family	Computing the shortest path		Computing the minimal paths	Computing the node-disjoint paths
	SFA	PCAACG	SFA	PCAACG
2D Torus	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Hypercube	$O(\log^3(n)\log(\log(n)))$	$O(\log(n))$	$O(\log(n)^4\log(\log(n)))$	$O(\log(n)^2)$
Bubble-sort		N/A		N/A
Star				
Transposition	$O(\log(n)^6\log(\log(n)))$		$O(\log(n)^8\log(\log(n)))$	
Butterfly	$O(1)$		$O(1)$	

Table 7: Time complexity of path computation algorithms on specific families of Cayley graphs.

Cayley graph family	Computing the shortest path		Computing the minimal paths	Computing the node-disjoint paths
	SFA	PCAACG	SFA	PCAACG
2D Torus	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Hypercube	$O(\log(n)^5)$	$O(\log(n)^{3/2})$	$O(\log(n)^5)$	$O(\log(n)^{5/2})$
Bubble-sort	$O(\log(n)^5)$	N/A	$O(\log(n)^{10})$	N/A
Star				
Transposition	$O(\log(n)^{10})$			
Butterfly	$O(1)$		$O(1)$	

failures. Then, the packet is returned to the source node, which forwards it to links that lead to paths without link failures. Hence the forwarding decision takes $O(D_\Gamma)$ time units. In this case, message headers consists of the source and destination labels and the path history of nodes traversed by the packet being routed, thus it can be represented by $O(D_\Gamma \log(n))$ bits.

The scheme was evaluated through computer simulations on a Borel CG of 1081 nodes in a random failure scenario, where the percentage of link failures increases from 5% to 35% [45]. Results show that the scheme is not shortest path and packet delivery is not guaranteed in the range of 1% to 32% of all pairs of source-destination nodes.

5.2.3 *Geometric routing with word-metric spaces*

The GRWMS applies techniques of word processing in groups presented in Chapter 4. Thus node labels are given by words in L, which can be represented by $O(D_\Gamma \log(\Delta_\Gamma))$ bits. The GRWMS follows a greedy routing strategy similar to the RPS, where message headers consist of the label of the destination node and the routing table of a node consists of the labels of its adjacent nodes. To forward packets, nodes computes the shortest paths from each of its adjacent nodes to the destination node. Then, packets are forwarded to the nearest node to the destination. The shortest paths are computed solving the MWP. As it is explained in Section 4.5, the MWP can be solved using the WD automaton, i.e. *Diff*, in time $O(|\mathbb{D}|\Delta_\Gamma|z|^2)$, where $z \leqslant 2D_\Gamma$ and $\mathbb{D}$ is the set of states of *Diff*. Therefore, the forwarding decisions are taken in time $O(|\mathbb{D}|\Delta_\Gamma^2 D_\Gamma^2)$ and the space complexity is given by the size of *Diff*, i.e. $|Diff|$.

5.2.4 *Comparison of routing schemes for Cayley graphs*

This section presents a comparison between the analyzed routing schemes working on the families of CGs presented in Table 2. As common features, the RPS and the GRWMS are deterministic and do not provide path diversity. In addition, they route packets through the shortest paths and guarantee the packet delivery. Conversely, the RCRR is fault-tolerant, do not compute the shortest path and do not guarantee packet delivery. These features are summarized in Table 8.

Table 8: Summary of routing schemes for Cayley graphs.

Routing scheme	Shortest path	Path diversity	Fault-tolerant	Packet delivery is guaranteed
RPS	Yes	No	No	Yes
RCRR	No		Yes	No
GRWMS	Yes		No	Yes

5.2.4.1 *Memory space requirements*

Table 9 presents the memory space requirements of the routing schemes previously presented. The table shows the space complexity of node labels, message headers, routing tables and extra information that would be required in the routing process. As it is explained above, the RCRR has the highest memory space requirements as it provides fault-tolerance and then requires further information about the global state of the network.

Table 9: Space complexity of routing schemes for Cayley graphs.

Routing scheme	Node label	Message header	Routing table	Other routing information
RPS	$O(p \log(p))$	$O(p \log(p))$	$O(\Delta_\Gamma p \log(p))$	N/A
RCRR	$O(\log(n))$	$O(D_\Gamma \log(n))$	$O(n\Delta_\Gamma)$	Dynamic routing table $O(n\Delta_\Gamma \log(D_\Gamma))$
GRWMS	$O(D_\Gamma \log(\Delta_\Gamma))$	$O(D_\Gamma \log(\Delta_\Gamma))$	$O(\Delta_\Gamma D_\Gamma \log(\Delta_\Gamma))$	Word-difference automaton $O(\lvert Diff \rvert)$

Regarding to the size of node labels and message headers, the RPS and GRWMS use the destination label as message headers, thereby both message headers and node label have the same space complexity. On the other hand, the RCRR adds the path history in the message header in order to provide fault-tolerance. Therefore, its message headers have size proportional to D_Γ. Table 10 compares the space complexity of node labels and message

headers defined by each routing scheme working on the evaluated CG. In all of them, the RCRR defines the smallest node labels as they consist of numbers from 0 to $n-1$. In contrast, the message headers defined by the RCRR are the largest ones due the message headers incorporate the path history. In the case of the RPS and the GRWMS, their node labels (and thus message headers) have the same size in Hypercube, Star and Transposition graphs. In Bubble-sort graphs, the node labels defined by the RPS are smaller than those defined by the GRWMS. Meanwhile in Butterfly graphs, the node labels defined by the GRWMS are smaller than those defined by the RPS.

Table 10: Space complexity of node labels and message headers defined by routing schemes on specific families of Cayley graph.

Cayley graph family	Node label / message header: RPS	Node label / message header: GRWMS	Node label: RCRR	Message header: RCRR
2D Torus	$O(n^{1/2}\log(n))$	$O(n^{1/2})$	$O(\log(n))$	$O(n^{1/2}\log(n))$
Hypercube	$O(\log(n)\log(\log(n)))$	$O(\log(n)\log(\log(n)))$		$O(\log(n)^2)$
Bubble-sort	$O(\log(n)\log(\log(n)))$	$O(\log(n)^2\log(\log(n)))$		$O(\log(n)^3)$
Star	$O(\log(n)\log(\log(n)))$	$O(\log(n)\log(\log(n)))$		$O(\log(n)^2)$
Transposition	$O(\log(n)\log(\log(n)))$	$O(\log(n)\log(\log(n)))$		$O(\log(n)^2)$
Butterfly	$O(\log(n)\log(\log(n)))$	$O(\log(n))$		$O(\log(n)^2)$

Turning now to the size of routing tables, the RPS and the GRWMS define routing tables consisting of the label of adjacent nodes, thereby their size

is proportional to the node degree. Conversely, the RCRR uses full routing tables, i.e. they contain an entry for each node in the CG, then their size is proportional to the number of nodes. In addition to the routing table, the RCRR and GRWMS use extra routing information. The RCRR uses a dynamic routing table to provide fault-tolerance and the GRWMS uses the automaton *Diff* to compute the shortest paths, see Section 4.5.

Table 11 compares the space complexity of the routing tables and extra routing information used by each routing scheme working on the evaluated families of CGs. Since the WPR extends the GRWMS, the values the size of *Diff* were empirically computed to evaluate the performance of the WPR. These values are presented in Table 19 and employed in Table 11. Considering the space requirements of both routing tables and extra routing information, the RPS has the lowest space complexity in all the evaluated CGs except by the 2D Torus graphs, where the GRWMS has the lowest space complexity. Conversely, the RCRR has the highest space complexity in all the evaluated CGs due to it uses full routing tables.

5.2.4.2 *Forwarding decision time*

Table 12 presents the forwarding decision time of the routing schemes introduced. In the case of the GRWMS, it depends on the values $|\mathbb{D}|$, which also determines the complexity of the WPR. An empirical estimation of $|\mathbb{D}|$ is presented in Table 17 and employed in Table 12. As it was mentioned, these schemes are deterministic and do not provide path diversity; in addition, the RCRR is fault-tolerant. According to this features, the forwarding decision time in deterministic mode of the three routing schemes was computed, meanwhile in fault-tolerant mode, the forwarding decision time was computed only for the RCRR.

In deterministic routing, the RCRR takes forwarding decisions making just a query to the routing table. Hence the forwarding decision is taken in constant time. In Fault-tolerant routing, the RCRR takes forwarding decision in time proportional to D_Γ when a packet that can not be forwarded due to failures, it is returned to the source node. In the worst case, said packet traverses $O(D_\Gamma)$ nodes before being forwarded through a path without failures.

On the other hand, the forwarding decision time in the RPS and the GRWMS is proportional to Δ_Γ as these schemes follows a greedy routing strategy, where the shortest path is computed Δ_Γ times. The computation of a shortest path takes time proportional to the size of a node label in the case of the RPS; and time proportional to the square of the size of a node label in the case of the GRWMS. Therefore the RPS takes forwarding decision faster than the GRWMS.

Table 13 compares the forwarding decision time of each routing scheme in specific families of CG. As it is expected, the GRWMS takes the slower forwarding decisions and the RCRR the fastest ones. On the other hand, the forwarding decisions are slow in the 2D Torus graphs than in the rest of the evaluated CG as its diameter grows faster than in the rest of CG. Conversely, the fastest forwarding decisions are taken in Butterfly graphs because of its diameter and node degree grow slower than in the rest of CG.

Table 11: Space complexity of routing tables (and other routing information) defined by routing schemes on specific families of Cayley graph.

Cayley graph family	Routing table			Dynamic routing table	Word-difference automaton
	RPS	RCRR	GRWMS	RCRR	GRWMS
2D Torus	$O(n^{1/2}\log(n))$	$O(n)$	$O(n^{1/2})$	$O(n\log(n))$	$O(1)$
Hypercube	$O(\log(n)^2\log(\log(n)))$	$O(n\log(n))$	$O(\log(n)^2\log(\log(n)))$	$O(n\log(n)\log(\log(n)))$	$O(\log(n)^3\log(\log(n)))$
Bubble-sort	$O(\log(n)^2\log(\log(n)))$	$O(n\log(n))$	$O(\log(n)^3\log(\log(n)))$	$O(n\log(n)\log(\log(n)))$	$O(\log(n)^5\log(\log(n)))$
Star	$O(\log(n)^2\log(\log(n)))$	$O(n\log(n))$	$O(\log(n)^2\log(\log(n)))$	$O(n\log(n)\log(\log(n)))$	$O(n\log(n)^2\log(\log(n)))$
Transposition	$O(\log(n)^3\log(\log(n)))$	$O(n\log(n)^2)$	$O(\log(n)^3\log(\log(n)))$	$O(n\log(n)^2\log(\log(n)))$	$O(\log(n)^6\log(\log(n)))$
Butterfly	$O(\log(n)\log(\log(n)))$	$O(n)$	$O(\log(n))$	$O(n\log(\log(n)))$	$O(n)$

Table 12: Forwarding decision time of routing schemes for Cayley graphs.

Routing scheme	Deterministic routing	Fault-tolerant routing
RPS	$O(\Delta_\Gamma p \log(p))$	N/A
RCRR	$O(1)$	$O(D_\Gamma)$
GRWMS	$O(\lvert\mathbb{D}\rvert \Delta_\Gamma^2 D_\Gamma^2)$	N/A

Table 13: Forwarding decision time of routing schemes on specific families of Cayley graph.

Cayley graph family	Deterministic routing		Fault-tolerant routing
	RPS	GRWMS	RCRR
2D Torus	$O(n^{1/2}\log(n))$	$O(n)$	$O(n^{1/2})$
Hypercube	$O(\log(n)^2 \log(\log(n)))$	$O(\log(n)^5)$	$O(\log(n))$
Bubble-sort	$O(\log(n)^2 \log(\log(n)))$	$O(\log(n)^9)$	$O(\log(n)^2)$
Star	$O(\log(n)^2 \log(\log(n)))$	$O(n \log(n)^4)$	$O(\log(n))$
Transposition	$O(\log(n)^3 \log(\log(n)))$	$O(\log(n)^7)$	$O(\log(n))$
Butterfly	$O(\log(n) \log(\log(n)))$	$O(n \log(n)^2)$	$O(\log(n))$
	RCRR: $O(1)$		

Part II

WORD-PROCESSING-BASED ROUTING

The Word-Processing-based Routing is a generic routing scheme for Cayley graphs, which applies word processing techniques for computing paths. The scheme has low time and space requirements, guarantees packet delivery, and provides: minimal routing, path diversity and fault-tolerance. The following chapters present the detailed definition of this routing scheme and its complexity analysis.

What though and care to determine the
exact site for a bridge, or for a fountain, and
to give a mountain road that perfect curve
which is at the same time the shortest...

— Marguerite Yourcenar

6

OVERVIEW

The WPR is a routing scheme designed to work on networks whose topology is defined by CGs of SAGs. It is considered the topology model presented Section 3.1, which works under the *asynchronous communication* model [19, Section 1.3.4]. The routing scheme is defined according to the model presented in Section 1.2.2.1, which applies the routing method called *store and forward* and operates from an approach of word processing in CGs presented in Chapter 4. Table 14 summarizes the basic notation used in the definition of the WPR.

Table 14: Notation of the Word-Processing-based Routing.

Parameter	**Definition**
$\mathcal{G} = \langle S \vert R \rangle$	Finitely presented group with generating set S and set of relators R.
$\Gamma(\mathcal{G}, S)$	Cayley graph of $\mathcal{G} = \langle S \vert R \rangle$ whose vertices $V(\Gamma)$ represent the nodes of a network and its edges $E(\Gamma)$ represent the links between nodes.
n	Number of nodes in $\Gamma(\mathcal{G}, S)$.
Δ_Γ	Regular degree of $\Gamma(\mathcal{G}, S)$.
D_Γ	Diameter of $\Gamma(\mathcal{G}, S)$.
$\mathcal{A}$	A lexicographically ordered alphabet, which satisfies Eq. (8) and defines the set of port labels in each node.
L	A language over $\mathcal{A}$, which satisfies Eq. (11) and defines the set of nodes labels.
$\overline{w}$	A node in $V(\Gamma)$ labeled as $w \in L$.
$\widehat{w}$	A path in $\Gamma(\mathcal{G}, S)$, where $w_{red} \in \mathcal{F}(\mathcal{A})$ denotes a sequence of output ports.
$\widehat{w}(\overline{u})$	A path in $\Gamma(\mathcal{G}, S)$, where $\overline{u}$ denotes the initial node and $w_{red} \in \mathcal{F}(\mathcal{A})$ denotes a sequence of output ports.
$\mathit{Diff} = (\mathbb{D}, \mathcal{B}, \delta, e_{\mathcal{A}})$	The WDA of $\Gamma(\mathcal{G}, S)$ with respect to $(\mathcal{A}, L)$, see Definition 4.4.3.

The WPR is composed of routing information, path computation algorithms, a fault-tolerant mechanism and forwarding protocols, which are presented in Chapter 7, Chapter 8 and Chapter 9, respectively. In addition, Chapter 10 presents a complexity analysis of the WPR. This chapter introduces the routing information used in each node and describes the general operation of the WPR and its process of node label assignment.

6.1 ROUTING INFORMATION

This section describes the routing information required in each node for maintaining the WPR.

6.1.1 *Routing table*

Nodes store a routing table that consists of only the port labels. Each port is labeled with a letter in $\mathcal{A}$ according to Eq. (8), then:

Preposition 6.1.1 *Every port label can be represented by* $O(\log(\Delta_\Gamma))$ *bits.*

6.1.2 *Node label*

Nodes are labeled with their *ShortLex* representative word in L according to Eq. (11). Every node in $V(\Gamma)$ labeled with $w \in L$ is referred to as $\overline{w}$.

Preposition 6.1.2 *For every node* $\overline{w}$*, its label* w *represents the ShortLex path from* $\overline{e_{\mathcal{A}}}$ *to* $\overline{w}$*, where* w *denotes a sequence of output ports.*

Lemma 6.1.1 *A node label consists of at most* D_Γ *letters in* $\mathcal{A}$*, thereby it can be represented by* $O(D_\Gamma \log(\Delta_\Gamma))$ *bits.*

Proof. It follows form Preposition 6.1.1 and Preposition 6.1.2. ■

6.1.3 *Word-difference automaton*

This automaton is used by the path computation algorithms due to it encodes the topological structure of the network, see Definition 4.4.3.

6.2 GENERAL OPERATION

The WPR allows the following properties:

1) Minimal routing.
2) Source routing.
3) Hop-by-hop routing.
4) Fault-tolerance.
5) Path diversity.

Further details about these properties can be found in Section 1.2.2. To support these properties the following path computation algorithms have been proposed:

- **Algorithm 4**: Compute the shortest path.
- **Algorithm 5**: Compute the minimal paths.
- **Algorithm 6**: Compute the paths of bounded length.
- **Algorithm 7**: Compute the K-shortest path.
- **Algorithm 8**: Compute the shortest link-disjoint paths.
- **Algorithm 9**: Compute the shortest node-disjoint paths.
- **Algorithm 10**: Compute the shortest paths avoiding a set of link and nodes.

These algorithms are formally presented in Chapter 7. Figure 14 shows which algorithms are used to provide each property of the WPR.

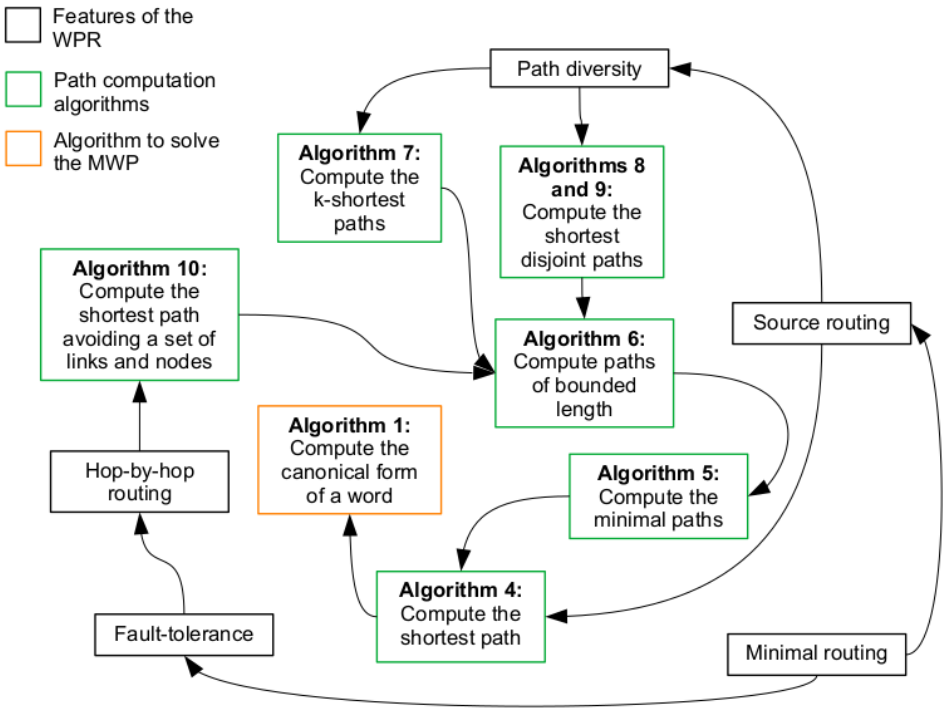


Figure 14: Relation between the properties of the WPR and their path computation algorithms. The arrows indicate algorithms supporting property and dependency between algorithms.

6.3 NODE LABEL ASSIGNMENT

This section describes the process to perform a node label assignment in a distributed way. First of all, ports of each node must be labeled with their corresponding letter in $\mathcal{A}$ according to Eq. (8). If its is unknown which generator is associated to each link, then an exploration technique such (as the one presented in Appendix A) must be performed in order to discover the path associated to the set of relators and derive the corresponding generator of each link.

After ports have been labeled, nodes must be labeled with their corresponding word in L according to Eq. (11). Algorithm 2 presents the steps to carry out the node label assignment in a distributed way. The label assignment begins in an arbitrary node r. This node is labeled with a special symbol $e_{\mathcal{A}}$, while the rest of nodes are labeled with *Null* (step 2). Then, r sends a message $\texttt{LABEL}(e_{\mathcal{A}})$ to its adjacent nodes. A node g has its turn to be labeled, when it receives a message LABEL for the first time, said message contains the label of the node h that sent the message. The label of g is the canonical form of the word resulting from concatenating the label of h with the label of the port that connects g with h. After g has been labeled, it sends a message LABEL to its adjacent nodes except to h. This message containing the label of g.

Algorithm 2 Distributed assignment of node labels.

1: Select an arbitrary node r of $V(\Gamma)$.
2: Initially, node r sets $\texttt{label} \leftarrow e_{\mathcal{A}}$. All other nodes set $\texttt{label} \leftarrow \textit{Null}$.
3: Every other node reacts to incoming messages as follows:

Upon receiving a message $\texttt{LABEL}(\texttt{label\_parent})$ through its port x **do**:

1: **if** $\texttt{label} = \textit{Null}$ **then**
2: $\quad\texttt{label} \leftarrow$ the canonical form of $\texttt{label\_parent} \cdot a$ (Algorithm 1).
3: $\quad$Send out the message $\texttt{LABEL}(\texttt{label})$ through all its ports in $\mathcal{A} \setminus \{x\}$.

Lemma 6.3.1 *The process of node label assignment for the WPR takes* $O(|\mathbb{D}|\Delta_\Gamma D_\Gamma^3)$ *time units and requires* $n(\Delta_\Gamma - 1) + 1$ *messages.*

Proof. Regarding to the time complexity. A node takes its turn to be labeled when it receives a message LABEL. Every node at distance l from node r receives a message $\texttt{LABEL}(\texttt{label\_parent})$, such that: $|\texttt{label\_parent}| = l - 1$ for $l > 1$, and $|\texttt{label\_parent}| = 1$ for $l = 1$. Then nodes at distance l from r take $O(|\mathbb{D}|\Delta_\Gamma l^2)$ time units in compute its node label due to Lemma 4.5.1. In addition, it is assumed that each message LABEL incurs in a delay of at most one time unit. Therefore, all nodes will have received a message LABEL and they have been labeled after $O(\sum_{l=2}^{D_\Gamma} |\mathbb{D}|\Delta_\Gamma l^2) = O(|\mathbb{D}|\Delta_\Gamma D_\Gamma^3)$ time units from the beginning of the execution. Turning now to the message complexity. Each node sends $\Delta_\Gamma + 1$ messages, except by r that sends Δ_Γ messages. Then the total number of messages transmitted during the execution of Algorithm 2 is $n(\Delta_\Gamma - 1) + 1$. ∎

How beautiful the world would be if
there were a rule for getting around in labyrinths.

— Umberto Eco

7 PATH COMPUTATION ALGORITHMS

This chapter presents the path computation algorithms used by the WPR. Assuming that the routing information has been configured as it was explained in Section 6.3, the path computation algorithms can be executed by any node.

7.1 PRELIMINARIES

Before presenting the path computation algorithms, it is necessary to explain how to compute: 1) paths recognized by the WDA and 2) the nodes and links of a path. These procedures are the base of the algorithms presented in this chapter.

7.1.1 *Paths recognized by the word-difference automaton*

From Definition 4.4.3, it follows immediately that:

Preposition 7.1.1 *If the WDA accepts* (r, t)*, such that* $r <_{\mathcal{A}} t$ *and* $q^{(r,t)} \in \mathbb{D}$*, then the following conditions holds:*

1. $|\widehat{r}| \leqslant |\widehat{t}|$,
2. *paths* $\widehat{r}$ *and* $\widehat{t}$*, beginning at the same node, are at uniform distance at most* k*, i.e.* $d_\Gamma(\widehat{r}, \widehat{t}) \leqslant k$ *(Lemma 4.4.1), and*
3. *the ShortLex path between the end nodes of* $\widehat{r}$ *and* $\widehat{t}$ *is* $\widehat{q^{(r,t)}}$.

These condition are illustrated in Figure 13, which shows two paths, in $BS(4)$, represented by a tuple of words accepted by the WDA.

Preposition 7.1.2 *If the WDA accepts* (r, t) *and* $q^{(r,t)}$*, then* $\widehat{r}$ *and* $t(\widehat{q^{(r,t)}})^{-1}$ *(beginning at the same node) are paths between the same pair of nodes due to* $r =_{\mathcal{G}} t(q^{(r,t)})^{-1}$.

Recall from Lemma 4.3.1 that words in the same equivalence class represent paths between the same pair of nodes, then:

Preposition 7.1.3 *If the WDA accepts* (r, t) *and* $q^{(r,t)} = e_{\mathcal{A}}$*, then*$r$ *and* t *are in the same equivalence class, i.e.* $r =_{\mathcal{G}} t$*, and* $\widehat{r}$ *and* $\widehat{t}$ *are paths between the same pair of nodes.*

Figure 15 illustrates the paths $\widehat{r}$, $\widehat{t}$ and $\widehat{q^{(r,t)}}$, where the tuple of words (r, t) is recognized by the WDA and the final state is $q^{(r,t)}$.

In the algorithms presented in this chapter, for a given path $\widehat{r}$, sometimes it is necessary to compute the set of words $t \in \mathcal{A}^*$ such that (r, t) is

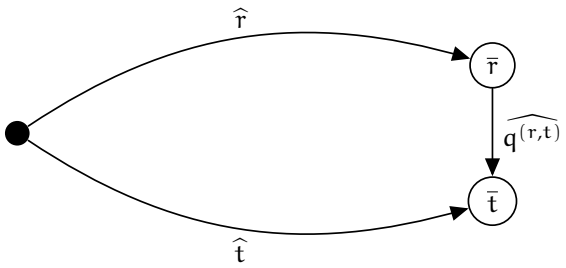


Figure 15: Geometric representation of a tuple of words (r, t) recognized by the word-difference automaton, such that the final state is $q^{(r,t)}$. The *Short-Lex* path between the end nodes of paths $\widehat{r}$ and $\widehat{t}$ is given by $\widehat{q^{(r,t)}}$.

recognized by WDA. Moreover, it is also useful to know the output state after reading each tuple (r, t), i.e. $q^{(r,t)}$. The following function gives this information.

$$\mathbb{T}(r) = \{(q, \mathcal{T}) \in \mathbb{D} \times \mathcal{A}^* : q = q^{(r,t)} \text{ iff } t \in \mathcal{T}\}, \tag{15}$$

this equation can be seen as a dictionary, where the keys are given by $\mathbb{D}$. The values of a key $q \in \mathbb{D}$ are the set of words $\mathcal{T}$ such that q is the state after reading any tuple in $r \times \mathcal{T}$.

From Preposition 7.1.2 and Preposition 7.1.3, every $\widehat{tq^{-1}}$, such that $(q, \mathcal{T}) \in \mathbb{T}(r)$ and $t \in \mathcal{T}$, is a path between the same nodes as $\widehat{r}$. For example, let $r = ac$ be a word representing a path in $BS(4)$. According to the WDA of $BS(4)$ (Table 4),

$$\mathbb{T}(ac) = \{(ac, \{\$\}), (c, \{a\}), (a, \{c\}), (cb, \{ab\}), (e_{\mathcal{A}}, \{ac, ca\}), (ab, \{cb\})\}, \tag{16}$$

and then words in the form tq^{-1} are given by the set

$$\{ac\$^{-1}, ca^{-1}, ac^{-1}, cb(ab)^{-1}, e_{\mathcal{A}}(ac)^{-1}, e_{\mathcal{A}}(ca)^{-1}, ab(bc)^{-1}\}. \tag{17}$$

These words represent paths between the same nodes as $\widehat{ac}$. Recall that for any $w \in \mathcal{A}^*$, $\widehat{w}$ is the path represented by the reduced form of w, i.e. w_{red} (see Definition 4.2.1). Hence words in Eq. (17) must be reduced by removing substrings of the form uu^{-1} and symbols $e_{\mathcal{A}}$ and \$, which represent the empty path. After reducing words, it is necessary to replace words of the form u^{-1} with words representing the inverse of u[1]. After reducing words, the resulting set is $\{ac, ca\}$ and thus paths $\widehat{ac}$ and $\widehat{ca}$ join the same pair of nodes.

Algorithm 3 computes $\mathbb{T}(r)$ implementing Eq. (15). The set $\mathbb{T}(r(i))$ is computed in each iteration of the *for* loop at line 2. The computation of $\mathbb{T}(r(i))$ is done by exploring all transitions (x_i, y) in $BS(4)$, for $r = x_1 \dots x_l$ and $y \in \mathcal{A}^+$, and keeping a track of words $t(i) = y_1 \dots y_i$, such that $q^{(r(i),t(i))} \in \mathbb{D}$. Figure 16 provides an illustration of Algorithm 3 with the WDA of $BS(4)$ and $r = ac$ as inputs.

Lemma 7.1.1 *Algorithm 3 computes the set of tuples* $\mathbb{T}(r)$ *in time* $O(|\mathbb{D}|\Delta_\Gamma|r|)$.

1 Recall that u^{-1} is given by the reverse string of the inverse letters in u, see Definition 4.2.1. For Bubble-sort graphs, the inverse of any letter $x \in \mathcal{A}$ is itself, i.e. $x^{-1} = x$

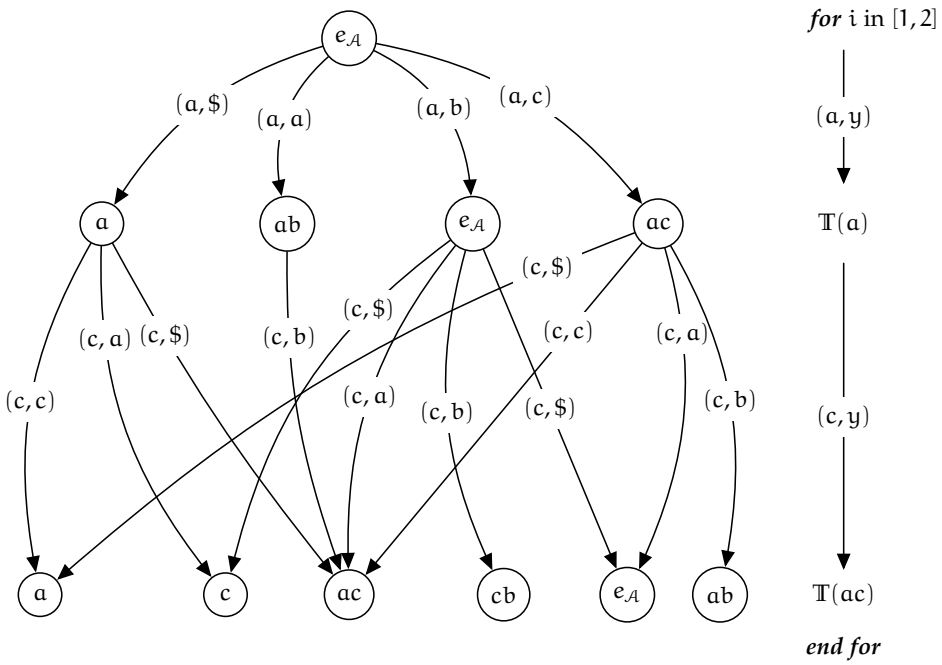


Figure 16: Illustration of the algorithm to compute paths recognized by the WDA (Algorithm 3). The inputs are the WDA of BS(4) (Table 4) and the word $r = ac$. The output is $\mathbb{T}(ac)$ (Eq. (16)).

Proof. For each $(q_{old}, \mathcal{T}_{old}) \in \mathbb{T}_{old}$, the set of transitions $\{q_{old}^{(x_i,y)} :\in A^+\}$ are computed. Since $|\mathbb{T}_{old}| \leqslant |\mathbb{D}|$, at most $|\mathbb{D}|(\Delta_\Gamma + 1)$ transition are computed. This process is repeated for each x_i in $r = x_1 \dots x_l$, then Algorithm 3 computes $\mathbb{T}(r)$ in time $O(|\mathbb{D}|\Delta_\Gamma|r|)$. ■

For a given word $r \in \mathcal{A}^*$ and state $q \in \mathbb{D}$, the set $\mathcal{T}$ is denoted by $\mathbb{T}(r, q)$, i.e.

$$\mathbb{T}(r, q) = \{t \in \mathcal{A}^* : q = q^{(r,t)} \text{ and } q \in \mathbb{D}\}. \tag{18}$$

Then $\widehat{tq^{-1}}$, for $t \in \mathbb{T}(r, q)$, are paths between the first and last nodes in $\widehat{r}$. For example, $\mathbb{T}(bacb, cb) = \{abac, babc\}$ according to Eq. (16). Then words in $\{abac(cb)^{-1}, babc(cb)^{-1}\}$ represent paths between the first and last nodes in $\widehat{bacb}$. After reducing the words, the resulting set is $\{abacbc, babcbc\}$ and then $\widehat{abacbc}$ and $\widehat{babcbc}$ are paths between the first and last nodes in $\widehat{bacb}$, see Figure 12b.

7.1.2 *Computing the links and nodes of a path*

Lemma 7.1.2 *Let* $E_{\widehat{r}}(\overline{u}) \subset L \times L$ *denote the set of links in the path* $\widehat{r}(\overline{u})$*, i.e.*

$$E_{\widehat{r}}(\overline{u}) = \{(e, f) \in L \times L : e =_{\mathcal{G}} ur(i) \text{ and } f =_{\mathcal{G}} ur(i+1), \text{ for } 0 < i < |r|\}. \tag{19}$$

The set $E_{\widehat{r}}(\overline{u})$ *can be computed in time* $O(|\mathbb{D}|\Delta_\Gamma|r|^3)$.

Proof. Recall from Lemma 4.5.1 that the canonical form a word $ur(i)$ is computed in time $O(|\mathbb{D}|\Delta_\Gamma(|u| + i)^2)$. Therefore, the canonical form of the words $ur(i)$ for $0 < i < |r|$ is computed in time $O(\sum_{i=1}^{|r|}(|\mathbb{D}|\Delta_\Gamma(|u| + i)^2)) = O(|\mathbb{D}|\Delta_\Gamma|r|^3)$. ■

Algorithm 3 Compute paths recognized by the word-difference automaton.

Input: The automaton $Diff = (\mathbb{D}, B, \delta, e_{\mathcal{A}})$.
Input: A word $r = x_1 \dots x_l \in \mathcal{A}^*$.
Output: The set of tuples $\mathbb{T}(r)$ representing words (r, t), where $t \in \mathcal{T}$, recognized by *Diff*.

$\mathbb{T}_{old} = \emptyset$, $\mathbb{T} = \{(e_{\mathcal{A}}, \emptyset)\}$.
for i in $[1 \dots l]$ **do**
  $\mathbb{T}_{old} \leftarrow \mathbb{T}$.
  $\mathbb{T} \leftarrow \emptyset$.
  for each $(q_{old}, \mathcal{T}_{old}) \in \mathbb{T}_{old}$ **do**
    for $y \in \mathcal{A}^+$ **do**
      $q_{new} \leftarrow q_{old}^{(x_i, y)}$
      if $q_{new} \in \mathbb{D}$ **then**
        if $\mathcal{T}_{old} = \emptyset$ OR $\mathcal{T}_{old} = \{\$\}$ **then**
          $\mathcal{T}_{new} \leftarrow \{y\}$.
        else
          $\mathcal{T}_{new} \leftarrow \{t \cdot y : t \in \mathcal{T}_{old}\}$.
        if $\exists (q, \mathcal{T}) \in \mathbb{T}$ such that $q = q_{new}$ **then**
          Replace $(q, \mathcal{T})$ by $(q, \mathcal{T} \cup \mathcal{T}_{new})$ in $\mathbb{T}$.
        else
          Add $(q_{new}, \mathcal{T}_{new})$ to $\mathbb{T}$.
return $\mathbb{T}$.

Lemma 7.1.3 *Let* $V_{\widehat{r}}(\overline{u}) \subset L$ *denote the set of intermediate nodes in the path* $\widehat{r}(\overline{u})$*, where the first and last nodes in the path are excluded, i.e.*

$$V_{\widehat{r}}(\overline{u}) = \{e \in L : e =_G ur(i), \text{ for } 1 < i < |r|\}. \tag{20}$$

The set $V_{\widehat{r}}(\overline{u})$ *can be computed in time* $O(|\mathbb{D}|\Delta_\Gamma|r|^3)$.

Proof. This proof is analogous to the proof of Lemma 7.1.2. ∎

Taking path $\widehat{ac}(\overline{b})$ in $BS(4)$ as example, $E_{\widehat{ac}}(\overline{b}) = \{(b, ba), (ba, bac)\}$ and $V_{\widehat{ac}}(\overline{b}) = \{ba\}$, see Figure 12b.

7.1.3 *Algorithm variables*

In addition to the notation introduced in Table 14, the path computation algorithms of the WPR employ the variables presented in Table 15.

Recall from Chapter 4 that:

- If $\mathcal{A}$ is an alphabet. Then $\mathcal{A}^*$ denotes all the words over $\mathcal{A}$, $\mathcal{F}(\mathcal{A}) \subset \mathcal{A}^*$ denotes all the reduced words over $\mathcal{A}$, and $L \subset \mathcal{F}(\mathcal{A})$ denotes the *ShortLex* words over $\mathcal{A}$.
- If $w \in L$. Then $\overline{w}$ denotes a node labeled as w, and $\widehat{w}$ denotes a *ShortLex* path defined by w_{red}.

Table 15: Variables of the path computation algorithms.

Variable	Definition
$k \in \mathbb{Z}^+$	The *fellow-traveler* constant of $\Gamma(\mathcal{G}, S)$.
$K \in \mathbb{Z}^+$	Number of computed paths.
$l \in \mathbb{Z}^+$	Length of the largest path computed.
$\overline{u}, \overline{v}$	Two nodes in $V(\Gamma)$.
$\widehat{w}$	Word representing the *ShortLex* path from $\overline{u}$ to $\overline{v}$.
$\mathbb{D}$	Set of WD of $\Gamma(\mathcal{G}, S)$ given by Eq. (14).

7.2 COMPUTING THE MINIMAL PATHS

7.2.1 *Computing the shortest path*

From Section 4.5, the problem of computing the shortest path between $\overline{u}$ and $\overline{v}$ is equivalent to computing the canonical form of the word $u^{-1}v$, i.e. $w \in L$ such that $w =_{\mathcal{G}} u^{-1}v$. The resulting word represents the *ShortLex* path between $\overline{u}$ and $\overline{v}$. For example, consider the nodes $\overline{abacba}$ and $\overline{bacba}$ in $BS(4)$. As it is shown in Example 4.5.1, the canonical form of the word $(abacba)^{-1}bacba =_{\mathcal{G}} abcababacba$ is $c \in L$ Eq. (13). Then $\widehat{c}$ is the *ShortLex* path from $\overline{abacba}$ to $\overline{bacba}$. Algorithm 4 computes the *ShortLex* path between $\overline{u}$ and $\overline{v}$ by computing the canonical form of $u^{-1}v$.

Algorithm 4 Compute the shortest path.

Input: The automaton $Diff = (\mathbb{D}, B, \delta, e_{\mathcal{A}})$.
Input: Label of the source node $u \in L$.
Input: Label of the destination node $v \in L$.
Output: The word $w \in L$ representing the *ShortLex* path between $\overline{u}$ and $\overline{v}$.

1: $z \leftarrow u^{-1}v$.
2: Compute the canonical form w of z (Algorithm 1).
3: **return** w.

Lemma 7.2.1 *Algorithm 4 computes the shortest path between two nodes of a* CG *in time* $O(|\mathbb{D}|\Delta_\Gamma D_\Gamma^2)$.

Proof. The canonical form of the word $u^{-1}v$ is computed once, which takes $O(|\mathbb{D}|\Delta_\Gamma|u^{-1}v|^2)$ time units, see Lemma 4.5.1. The length of words u and v is $O(D_\Gamma)$ due to Lemma 6.1.1, then Algorithm 4 runs in time $O(|\mathbb{D}|\Delta_\Gamma D_\Gamma^2)$. ■

7.2.2 *Computing the minimal paths*

An important consequence of the k-*fellow-traveler* property (Lemma 4.4.1) is that minimal paths in CGs of SAGs are separated uniformly at distance at most k [25, Lemma 3.2.3]. From this results, this section presents a method to compute all minimal paths between two nodes. The method consists in

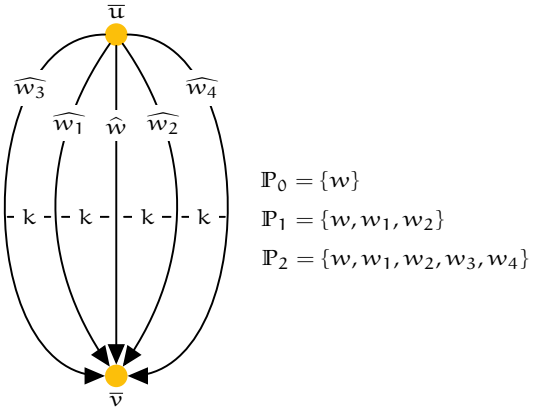


Figure 17: Minimal paths between two nodes of a Cayley graph of a *ShortLex* automatic group. The set $\mathbb{P}_j(w)$ (Eq. (21)) is given by the words at distance at most ik from the *ShortLex* path $\widehat{w}$.

computing *ShortLex* path first, and then the paths at uniform distance at most ik, for $1 \leqslant i \leqslant D_\Gamma$, from the *ShortLex* path.

Definition 7.2.1 *Let $\mathbb{P}_j(w) \subset \mathcal{F}(\mathcal{A})$ be the set of words representing the minimal paths between $\overline{u}$ and $\overline{v}$ that are at distance at most jk from $\widehat{w}$, i.e.*

$$\mathbb{P}_j(w) = \{t \in \mathcal{F}(\mathcal{A}) : t =_\mathcal{G} w, |t| = |w| \text{ and } d_\Gamma(\widehat{t}, \widehat{w}) \leqslant jk\}, \tag{21}$$

where $j \geqslant 0$.

Figure 17 provides a geometric representation of paths defined by words in Eq. (21). As it is shown, for any path $\widehat{w}$, where $w \in \mathbb{P}_j$, it exists a path $\widehat{r}$, where $r \in \mathbb{P}_{j-1}$, such that $\widehat{w}$ and $\widehat{r}$ follow the k-*fellow-traveler* property and then they are accepted by the WDA, where the output state is $e_\mathcal{A}$, i.e. $q^{(r,t)} = e_\mathcal{A}$. Therefore:

Lemma 7.2.2 *The set $\mathbb{P}_j(w)$ can be built recursively as follows:*

$$\mathbb{P}_j(w) = \{t \in \mathcal{F}(\mathcal{A}) : \exists r \in \mathbb{P}_{j-1}(w) \text{ s.t. } q^{(r,t)} = e_\mathcal{A}\}, \tag{22}$$

where $j > 0$ and $\mathbb{P}_0(w) = \{w\}$.

Proof. The proof is by induction on $\mathbb{P}_j(w)$. Clearly, two paths are at distance 0 if and only if they are the same path. So, $\mathbb{P}_0(w) = \{w\}$. Assuming Eq. (22) holds for $\mathbb{P}_{j-1}(w)$, it must be proved it for $\mathbb{P}_j(w)$. Eq. (22) implies that: 1) $t =_\mathcal{G} w$ due to $t =_\mathcal{G} r$ (by Preposition 7.1.3) and $r =_\mathcal{G} w$ (by the induction hypothesis); 2) $|t| = |w|$ due to $|t| = |r|$ (by Definition 4.4.3) and $|r| = |w|$ (by the induction hypothesis); and 3) by the triangle inequality $d_\Gamma(\widehat{t}, \widehat{w}) \leqslant jk$ due to $d_\Gamma(\widehat{r}, \widehat{t}) \leqslant k$ (by Preposition 7.1.1) and $d_\Gamma(\widehat{r}, \widehat{w}) \leqslant (j-1)k$ (by the induction hypothesis). Therefore Eq. (22) defines the shortest paths between $\overline{u}$ and $\overline{v}$ that are at distance at most jk from $\widehat{w}$ and Lemma 7.2.2 holds for $j \geqslant 0$. ■

Example 7.2.1 *Consider the nodes $\overline{aba}$ and $\overline{cb}$ in $BS(4)$ (Figure 12b). From the previous section, the ShortLex path from $\overline{aba}$ to $\overline{cb}$ is defined by the canonical form of $(aba)^{-1}cb$, i.e. $abacb \in L$ (Eq. (13)). According to Eq. (22) and the WDA of $BS(4)$ (Table 4), $\mathbb{P}_0(abacb) = \{abacb\}$ and:*

$$\mathbb{P}_1(abacb) = \{abacb, abcab, bacbc\},$$

$$\mathbb{P}_2(abacb) = \{abacb, abcab, bacbc, bcabc\},$$

$$\mathbb{P}_3(abacb) = \{abacb, abcab, bacbc, bcabc\}.$$

Note that $\mathbb{P}_3(abacb) = \mathbb{P}_2(abacb)$, *then* $\mathbb{P}_i(abacb) = \mathbb{P}_2(abacb)$, *for any* $i > 2$. *Therefore, words in* $\mathbb{P}_2(abacb)$ *represent all the minimal paths from* $\overline{aba}$ *to* $\overline{cb}$. *In addition, words in* $\mathbb{P}_3(abacb)$ *represent paths at distance* 6 *from* $\widehat{abacb}$ *as* $k = 2$ *for* BS(4). *Since* $D_\Gamma = 6$ *for* BS(4), *words in* $\mathbb{P}_3(abacb)$ *represent all the minimal paths from* $\overline{aba}$ *to* $\overline{cb}$.

Corollary 7.2.1 *Words in* $\mathbb{P}_{\lceil D_\Gamma/k \rceil}(w)$ *represent all the minimal paths between* $\overline{u}$ *and* $\overline{v}$ *due to* $d_\Gamma(\widehat{w}, \widehat{p}) \leqslant D_\Gamma$ *for any path* $\widehat{p}$ *from* $\overline{u}$ *to* $\overline{v}$.

Given the labels of two nodes $u, v \in L$, Algorithm 5 computes a set of the minimal paths from $\overline{u}$ to $\overline{v}$. First, the *ShortLex* path is computed (line 2). Then, the set $\mathbb{P}_j(w)$ is sequentially computed implementing Eq. (22) for $j > 1$ as follows: for each word $r \in \mathbb{P}_j(w) \setminus \mathbb{P}_{j-1}(w)$, the set of words $t \in \mathcal{F}(\mathcal{A})$ such that $q^{(r,t)} = e_\mathcal{A}$, i.e. $\mathcal{T}_{red}$, is computed (lines 6-7). The set $\mathbb{P}_j(w)$ results from the union of all $\mathcal{T}_{red}$ (lines 8-9). This process is repeated until all the minimal paths have been computed, i.e. $j = \lceil \frac{D_\Gamma}{k} \rceil$, (*while* loop at line 4).

Algorithm 5 Compute the minimal paths.

Input: The automaton $Diff = (\mathbb{D}, B, \delta, e_\mathcal{A})$.
Input: Label of the source node $u \in L$.
Input: Label of the destination node node $v \in L$.
Input: Value of the network diameter D_Γ.
Input: Value of the *fellow-traveler* constant k.
Output: The set of words $\mathbb{P}_j(w) \subset \mathcal{F}(\mathcal{A})$ representing a set of the minimal paths between from $\overline{u}$ to $\overline{v}$.

1: $z \leftarrow u^{-1}v$, $j \leftarrow 1$, $\mathbb{P}_{i-1} \leftarrow \emptyset$, $\mathbb{P}_{temp} \leftarrow \emptyset$.
2: Computes the canonical form w of z (Algorithm 1).
3: $\mathbb{P}_i \leftarrow \{w\}$.
4: **while** $\mathbb{P}_i \neq \mathbb{P}_{i-1}$ AND $j \leqslant \left\lceil \frac{D_\Gamma}{k} \right\rceil$ **do**
5: **for** $r \in \mathbb{P}_i \setminus \mathbb{P}_{i-1}$ **do**
6: Compute $\mathbb{T}(r)$ (Algorithm 3).
7: $\mathcal{T}_{red} \leftarrow \{t_{red} \in \mathcal{F}(\mathcal{A}) : t \in \mathbb{T}(r, e_\mathcal{A})\}$.
8: $\mathbb{P}_{temp} \leftarrow \mathbb{P}_{temp} \cup \mathcal{T}_{red}$.
9: $j \leftarrow j + 1$, $\mathbb{P}_{i-1} \leftarrow \mathbb{P}_i$, $\mathbb{P}_i \leftarrow \mathbb{P}_i \cup \mathbb{P}_{temp}$.
10: **return** $\mathbb{P}_i$.

Lemma 7.2.3 *Algorithm 5 computes the set of the minimal paths* $\mathbb{P}_j(w)$ *between any two nodes of a* CG *in time* $O(\max\{|\mathbb{P}_{j-1}(w)|, D_\Gamma\}|\mathbb{D}|\Delta_\Gamma D_\Gamma)$.

Proof. The canonical form of $u^{-1}v$ is computed in time $O(|\mathbb{D}|\Delta_\Gamma D_\Gamma^2)$ due to Lemma 4.5.1. Then the set $\mathbb{P}_j(w)$ is computed for $0 < j < \left\lceil \frac{D_\Gamma}{k} \right\rceil$. It implies that the set $\mathbb{T}(r)$ is computed $|\mathbb{P}_{j-1}(w)|$ times. The computation of $\mathbb{T}(r)$ takes $O(|\mathbb{D}|\Delta_\Gamma|r|)$ time units due to Lemma 7.1.1. Since $|r| = |w|$ and $|\widehat{w}| \leqslant D_\Gamma$, Algorithm 5 computes $\mathbb{P}_j(w)$ in time $O(\max\{|\mathbb{P}_{j-1}(w)|, D_\Gamma\}|\mathbb{D}|\Delta_\Gamma D_\Gamma)$. ■

7.3 COMPUTING THE k-SHORTEST PATHS

7.3.1 *Computing paths of bounded length*

Definition 7.3.1 *Let* $[w]_i \subset \mathcal{F}(\mathcal{A})$ *be the set of words representing the paths between* $\overline{u}$ *and* $\overline{v}$ *whose length is at most* $|w| + ik$*, i.e.*

$$[w]_i = \{p \in \mathcal{F}(\mathcal{A}) : p =_{\mathcal{G}} w \text{ and } |\widehat{p}| \leqslant |\widehat{w}| + ik\}, \tag{23}$$

where $i \geqslant 0$.

Theorem 7.3.1 *The set* $[w]_i$ *can be built recursively as follows:*

$$\begin{aligned}[w]_i = \{&p \in \mathcal{F}(\mathcal{A}) : p = (t(q^{(r,t)})^{-1})_{red},\\ &\exists r \in [w]_{i-1} \text{ and } t \in \mathcal{F}(\mathcal{A}) \text{ s.t. } q^{(r,t)} \in \mathbb{D}\},\end{aligned} \tag{24}$$

where $i > 0$ *and* $[w]_0 = \mathbb{P}_{\lceil D_\Gamma/k \rceil}(w)$.

Proof. The proof is by induction on $[w]_i$. By Corollary 7.2.1, $[w]_0 = \mathbb{P}_{\lceil D_\Gamma/k \rceil}(w)$. Assuming Eq. (24) to hold for $[w]_{i-1}$, it must be proved it for $[w]_i$. Eq. (24) implies that: 1) $p =_{\mathcal{G}} w$ due to $p =_{\mathcal{G}} r$ (by Preposition 7.1.2) and $r =_{\mathcal{G}} w$ (by the induction hypothesis); and 2) $|\widehat{p}| \leqslant |\widehat{w}| + ik$ due to $|p| = |t| + |q^{(u,v)}|$, where $|q^{(u,v)}| \leqslant k$, $|t| = |r|$ (by Definition 4.4.3), and $|r| \leqslant |w| + (i-1)k$ (by the induction hypothesis). Therefore, Eq. (24) defines the set of paths $\widehat{p}$ between $\overline{u}$ and $\overline{v}$ that satisfy $|\widehat{p}| \leqslant |\widehat{w}| + ik$ and Theorem 7.3.1 holds for $i \geqslant 0$. ■

Corollary 7.3.1 *Let* i *be the smallest positive integer such that* $[w]_i = [w]_{i-1}$. *Then, words in* $[w]_{i-1}$ *represent all paths between* $\overline{u}$ *and* $\overline{v}$.

Corollary 7.3.2 *The set of words* p *representing the paths between* $\overline{u}$ *and* $\overline{v}$ *such that* $|\widehat{w}| + (i-2)k < |\widehat{p}| \leqslant |\widehat{w}| + ik$ *is given by:*

$$\begin{aligned}[w]_i \setminus [w]_{i-2} = \{&p \in \mathcal{F}(\mathcal{A}) : p = (t(q^{(r,t)})^{-1})_{red},\\ &\exists\, r \in [w]_{i-1} \setminus [w]_{i-2} \text{ and } t \in \mathcal{F}(\mathcal{A}) \text{ s.t. } q^{(r,t)} \in \mathbb{D}\}.\end{aligned} \tag{25}$$

Given the set $[w]_{i-1}$, Algorithm 6 computes $[w]_i$. First, $[w]_{i-2}$ is computed (line 1). Then, $[w]_i$ is initialized to $[w]_{i-2}$ (line 2). Finally, Eq. (25) is implemented (*for* loop at line 3). The set $[w]_i$ results from the union of $[w]_{i-2}$ and $[w]_i \setminus [w]_{i-2}$.

Lemma 7.3.1 *Given the set* $[w]_{i-1}$, *Algorithm 6 computes the paths of length bounded by* $l = |w| + ik$, *i.e.* $[w]_i$, *in time* $O(|\mathbb{D}|\Delta_\Gamma l|[w]_{i-1} \setminus [w]_{i-2}|)$.

Proof. The set $\mathbb{T}(r)$ is computed $|[w]_{i-1} \setminus [w]_{i-2}|$ times. The computation of $\mathbb{T}(r)$ takes $O(|\mathbb{D}|\Delta_\Gamma|r|)$ time units due to Lemma 7.1.1. From Eq. (21), $|r| \leqslant |w| + ik = l$. Therefore, Algorithm 6 computes $[w]_i$ in time $O(|\mathbb{D}|\Delta_\Gamma l|[w]_{i-1} \setminus [w]_{i-2}|)$. ■

Example 7.3.1 *Consider the set of words* $[bac]_0 = \{bac, bca\}$ *and the WDA of* $BS(4)$ *(Table 4). To compute* $[bac]_1$, *Algorithm 6 proceeds as follows:*

1) $[bac]_{-1} \leftarrow \emptyset$.

2) $[bac]_1 \leftarrow \emptyset$.

Algorithm 6 Compute paths of bounded length.

Input: The automaton $\mathit{Diff} = (\mathbb{D}, B, \delta, e_{\mathcal{A}})$.
Input: The set of words $[w]_{i-1} \subset \mathcal{F}(\mathcal{A})$.
Output: The set of words $[w]_i \subset \mathcal{F}(\mathcal{A})$ representing the paths of length bounded by $|w| + ik$ between any pair of nodes $\overline{u}$ and $\overline{v}$.

1: $[w]_{i-2} \leftarrow \{r \in [w]_{i-1} : |r| \leqslant |w| + (i-2)k\}$.
2: $[w]_i \leftarrow [w]_{i-2}$.
3: **for** $r \in [w]_{i-1} \setminus [w]_{i-2}$ **do**
4: Compute $\mathbb{T}(r)$ (Algorithm 3).
5: $\mathcal{U}_r \leftarrow \{(t(q)^{-1})_{red} \in \mathcal{F}(\mathcal{A}) : t \in \mathcal{T}, \forall (q, \mathcal{T}) \in \mathbb{T}(r)\}$.
6: $[w]_i \leftarrow [w]_i \cup \mathcal{U}_r$.
7: **return** $[w]_i$.

3) $r \leftarrow bac$
   a) $\mathbb{T}(r) \leftarrow \{(e_{\mathcal{A}}, \{bca\}), (cb, \{aba\})\}$.
   b) $\mathcal{U}_r \leftarrow \{bca, ababc\}$.
   c) $[bac]_1 \leftarrow \emptyset \cup \{bca, ababc\}$.

4) $r \leftarrow bca$
   a) $\mathbb{T}(r) \leftarrow \{(e_{\mathcal{A}}, \{bac\}), (ab, \{cbc\})\}$.
   b) $\mathcal{U}_r \leftarrow \{bac, cbcba\}$.
   c) $[bca]_1 \leftarrow \{bca, ababc\} \cup \{bac, cbcba\}$.

5) *Return* $[bca]_1$.

Therefore, $\widehat{bac}$, $\widehat{bca}$, $\widehat{ababc}$ *and* $\widehat{cbcba}$ *are all the paths in* BS(4) *(see Figure 12b) satisfying the following conditions: 1) they join the same nodes as* $\widehat{bac}$*, and 2) their length bounded by* $ik + |bac|$*, where* $i = 1$ *and* $k = 2$.

7.3.2 *Computing the K-shortest paths*

Algorithm 7 computes the K-shortest paths between two nodes $\overline{u}$ and $\overline{v}$. First, the set of the minimal paths $\mathbb{P}_{\lceil D_\Gamma/k \rceil}$ is computed (line 2). Then, the set of paths $[w]_i$ is sequentially computed for $i \geqslant 2$ and stored in $[w]_{new}$ (line 12-13). The algorithm finishes when the K-shortest paths have been computed, i.e. $|[w]_{new}| \geqslant K$ (*if* condition at line 5) or when all possible paths haven computed, i.e. $[w]_{new} = [w]_{old}$ (*while* loop at line 4).

Lemma 7.3.2 *Algorithm 7 computes the* K*-shortest paths, where* $K > 1$*, between two nodes of a* CG *in time* $O(\max\{Kl, D_\Gamma^2\}|\mathbb{D}|\Delta_\Gamma)$*, where* l *is the length of the* K*-th shortest path.*

Proof. The set of the minimal paths is computed in time $O(\max\{|\mathbb{P}_{\lceil D_\Gamma/k \rceil - 1}, D_\Gamma\}|\mathbb{D}|\Delta_\Gamma D_\Gamma)$ due to Lemma 7.2.3. Then the set $[w]_i$ is computed for $1 \leqslant i \leqslant K$. It takes $O(|\mathbb{D}|\Delta_\Gamma l|[w]_{i-1}|)$ time units due to Lemma 7.3.1. Since $\mathbb{P}_{\lceil D_\Gamma/k \rceil - 1} \subset [w]_{i-1}$ and $K \approx |[w]_{i-1}|$, the algorithm finishes in time $O(\max\{Kl, D_\Gamma^2\}|\mathbb{D}|\Delta_\Gamma)$. ∎

Example 7.3.2 *Consider the nodes* $\overline{ac}$ *and* $\overline{abcba}$ *in* BS(4) *(Figure 12b), and the integer* $K = 3$*. To compute the 3-shortest paths, Algorithm 7 proceeds as follows:*

Algorithm 7 Compute the K-shortest paths.

Input: Label of the source node $u \in L$.
Input: Label of the destination node $v \in L$.
Input: A positive integer K.
Input: Value of the network diameter D_Γ.
Input: Value of the *fellow-traveler* constant k.
Output: A set of words $\mathcal{P} \subset \mathcal{F}(\mathcal{A})$ representing the K-shortest paths from $\overline{u}$ to $\overline{v}$.

1: $i \leftarrow 1$, $[w]_{old} \leftarrow \emptyset$, $\mathcal{P} \leftarrow \emptyset$.
2: Compute the minimal paths $\mathbb{P}_{\lceil D_\Gamma/k \rceil}(w)$ (Algorithm 5).
3: $[w]_{new} \leftarrow \mathbb{P}_{\lceil D_\Gamma/k \rceil}(w)$.
4: **while** $([w]_{new} \neq [w]_{old})$ **do**
5: **if** $|[w]_{new}| \geqslant K$ **then**
6: $\mathcal{P} \leftarrow$ Set of the first K-*ShortLex* words in $[w]_{new}$.
7: **return** $\mathcal{P}$.
8: **else**
9: Compute $[w]_i$ (Algorithm 6).
10: $[w]_{old} \leftarrow [w]_{new}$, $[w]_{new} \leftarrow [w]_i$, $\mathcal{P} \leftarrow [w]_i$, $i \leftarrow i + 1$.
11: **return** $\mathcal{P}$.

1) $i \leftarrow 1$, $[w]_{old} \leftarrow \emptyset$, $\mathcal{P} \leftarrow \emptyset$.

2) $\mathbb{P}_3(w) \leftarrow \{bac, bca\}$ *(in* $BS(4)$, $k = 2$ *and* $D_\Gamma = 6$*).*

3) $[w]_{new} \leftarrow \{bac, bca\}$.

4) $[w]_{new} = \{bac, bca\}$ *and* $[w]_{old} = \emptyset$*, then*

a) $|[w]_{new}| = 2$ *is less than* $K = 3$*, then*

i. $[w]_1 \leftarrow \{bac, bca, ababc, cbcba\}$.

ii. $[w]_{old} \leftarrow \{bac, bca\}$, $[w]_{new} \leftarrow \{bac, bca, ababc, cbcba\}$, $\mathcal{P} \leftarrow \{bac, bca, ababc, cbcba\}$, $i \leftarrow 2$.

5) $[w]_{new} = \{bac, bca, ababc, \}$ *and* $[w]_{old} = \{bac, bca\}$*, then*

a) $|[w]_{new}| = 4$ *is greater than* $K = 3$*, then*

i. $\mathcal{P} \leftarrow \{bac, bca, ababc\}$.

ii. Return $\mathcal{P}$.

Therefore, the 3-shortest paths from $\overline{ac}$ *to* $\overline{abcba}$ *are* $\widehat{bac}(\overline{ac})$, $\widehat{bca}(\overline{ac})$ *and* $\widehat{abacbc}(\overline{ac})$.

7.4 COMPUTING THE DISJOINT PATHS

7.4.1 *Computing the shortest link-disjoint paths*

Algorithm 8 computes the shortest link-disjoint paths two nodes $\overline{u}$ and $\overline{v}$, i.e. $\mathcal{D}_E(u, v)$, where $|\mathcal{D}_E(u, v)| = \Delta_\Gamma$ as CG are node-transitive [28, Theorem 3.7]. First, the set of the minimal paths $\mathbb{P}_{\lceil D_\Gamma/k \rceil}$ is computed (line 2). Then, the set of paths $[w]_i$ is sequentially computed for $i \geqslant 2$ and stored in $[w]_{new}$ (line 12-13). The shortest link-disjoint path from paths in $\mathcal{D}_E(u, v)$

is searched in $[w]_i \setminus [w]_{i-1}$ (*for* loop at line 5). The strategy to identify link-disjoint paths is to keep a record of the links of paths in $\mathcal{D}_E(u,v)$, which is given by the set $E_D(\overline{u})$. For each $r \in [w]_i \setminus [w]_{i-1}$, the intermediate links in the path $\widehat{r}(\overline{u})$ are computed, i.e. $E_{\widehat{r}}(\overline{u})$ (line 6). If $E_D(\overline{u}) \cap E_{\widehat{r}}(\overline{u}) = \emptyset$, then $\widehat{r}$ is link-disjoint from paths in $\mathcal{D}_E(u,v)$ (*if* condition at line 7). Thereby r is added to $\mathcal{D}_E(u,v)$ and $E_D(\overline{u})$ is updated (lines 8-9). In addition, it is checked if all link-disjoint paths have been computed, if so $\mathcal{D}_E(u,v)$ is returned (*if* condition at line 10). Otherwise, this process is repeated for $i+1$, and so on until $|\mathcal{D}_E(u,v)| = \Delta_\Gamma$ (*while* loop at line 4).

Algorithm 8 Compute the shortest link-disjoint paths.

Input: Label of the source node $u \in L$.
Input: Label of the destination node $v \in L$.
Input: Value of the network degree Δ_Γ.
Input: Value of the network diameter D_Γ.
Input: Value of the *fellow-traveler* constant k.
Output: A set of words $\mathcal{D}_E \subset \mathcal{F}(\mathcal{A})$ representing the shortest link-disjoint paths from $\overline{u}$ to $\overline{v}$.

1: $i \leftarrow 1$, $[w]_{old} \leftarrow \emptyset$, $\mathcal{D}_E \leftarrow \emptyset$, $E_D \leftarrow \emptyset$.
2: Compute the minimal paths $\mathbb{P}_{\lceil D_\Gamma/k \rceil}(w)$ (Algorithm 5).
3: $[w]_{new} \leftarrow \mathbb{P}_{\lceil D_\Gamma/k \rceil}(w)$.
4: **while** $([w]_{new} \neq [w]_{old})$ **do**
5: **for** $r \in [w]_{new} \setminus [w]_{old}$ **do**
6: Compute $E_{\widehat{r}}$ (Lemma 7.1.2).
7: **if** $E_D \cap E_{\widehat{r}} = \emptyset$ **then**
8: Add r to $\mathcal{D}_E$.
9: $E_D \leftarrow E_D \cup E_{\widehat{r}}$.
10: **if** $|\mathcal{D}_E| = \Delta_\Gamma$ **then**
11: **return** $\mathcal{D}_E$.
12: Compute $[w]_i$ (Algorithm 6).
13: $[w]_{old} \leftarrow [w]_{new}$, $[w]_{new} \leftarrow [w]_i$, $i \leftarrow i+1$.
14: **return** $\mathcal{D}_E$.

Lemma 7.4.1 *Algorithm 8 computes the shortest link-disjoint paths between any two nodes of a CG in time $O(k|\mathbb{D}|\Delta_\Gamma l^3)$, where the longest path computed has length l and is the k-th shortest path.*

Proof. The set of the minimal paths is computed in time $O(\max\{|\mathbb{P}_{\lceil D_\Gamma/k \rceil - 1}, D_\Gamma\}|\mathbb{D}|\Delta_\Gamma D_\Gamma)$ due to Lemma 7.2.3. Then the set $[w]_i$ is computed for $1 \leqslant i \leqslant k$. It takes $O(|\mathbb{D}|\Delta_\Gamma l|[w]_{i-1}|)$ time units due to Lemma 7.3.1. In addition, for each $r \in [w]_i$, the set $E_{\widehat{r}}(\overline{u})$ is computed in time $O(|\mathbb{D}|\Delta_\Gamma l^3)$ due to Lemma 7.1.2. Since $k \approx |[w]_i|$, Algorithm 8 finishes in time $O(k|\mathbb{D}|\Delta_\Gamma l^3)$. ∎

Example 7.4.1 *To compute the shortest link-disjoint paths from $\overline{c}$ to $\overline{ac}$ in BS(4) (Figure 12b), Algorithm 8 proceeds as follows:*

1) $i \leftarrow 1$, $[w]_{old} \leftarrow \emptyset$, $\mathcal{D}_E \leftarrow \emptyset$, $E_D \leftarrow \emptyset$.

2) $\mathbb{P}_3(w) \leftarrow \{a\}$ *(in* BS(4), $k = 2$ *and* $D_\Gamma = 6$*).*

3) $[w]_{new} \leftarrow \{a\}$.

4) $[w]_{new} = \{a\}$ *and* $[w]_{old} = \emptyset$, *then*
   a) $r \leftarrow a$
      i. $E_{\widehat{a}}(\overline{c}) \leftarrow [(c, ac)]$.
      ii. $E_D \cap E_{\widehat{c}}(\overline{a}) = \emptyset$, *then*
         - $\mathcal{D}_E \leftarrow \{a\}$.
         - $E_D \leftarrow [(c, ac)$.
         - $|\mathcal{D}_E| = 1$ *is less than* $\Delta_\Gamma = 3$.
   b) $[w]_1 \leftarrow \{a, cac\}$.
   c) $[w]_{old} \leftarrow \{a\}$, $[w]_{new} \leftarrow \{a, cac\}$, $i \leftarrow 2$.
   d) $r \leftarrow cac$.
      i. $E_{\widehat{cac}}(\overline{c}) \leftarrow \{(c, e_{\mathcal{A}}), (e_{\mathcal{A}}, a), (a, ac)\}$.
      ii. $E_D \cap E_{\widehat{cac}}(\overline{c}) = \emptyset$, *then*
         - $\mathcal{D}_E \leftarrow \{a, cac\}$.
         - $E_D \leftarrow \{(c, ac), (c, e_{\mathcal{A}}), (e_{\mathcal{A}}, a), (a, ac)\}$.
         - $|\mathcal{D}_E| = 2$ *is less than* $\Delta_\Gamma = 3$.
   e) $[w]_2 \leftarrow \{a, cac, babab\}$.
   f) $[w]_{old} \leftarrow \{a, cac\}$, $[w]_{new} \leftarrow \{a, cac, babab\}$, $i \leftarrow 3$.
   g) $r \leftarrow babab$.
      i. $E_{\widehat{babab}}(\overline{c}) \leftarrow \{(c, cb), (cb, cba), (cba, acba), (acba, acb), (acb, ac)\}$.
      ii. $E_D \cap E_{\widehat{babab}} = \emptyset$, *then*
         - $\mathcal{D}_E \leftarrow \{a, cac, babab\}$.
         - $E_D \leftarrow \{(c, ac), (c, e_{\mathcal{A}}), (e_{\mathcal{A}}, a), (a, ac), (c, cb), (cb, cba), (cba, acba), (acba, acb), (acb, ac)\}$.
         - $|\mathcal{D}_E| = \Delta_\Gamma$, *then*
            – *Return* $\mathcal{D}_E = \{a, cac, babab\}$.

Therefore, the shortest link-disjoint paths from $\overline{a}$ *to* $\overline{ac}$ *are* $\widehat{a}(\overline{a})$, $\widehat{cac}(\overline{a})$ *and* $\widehat{babab}(\overline{a})$.

7.4.2 *Computing the shortest node-disjoint paths*

Similarly to Algorithm 8, Algorithm 9 computes the set of node-disjoint paths between any pair of nodes $\overline{u}$ and $\overline{v}$, i.e. $\mathcal{D}_V(u, v)$, where $\frac{2}{3}(\Delta + 1) \leqslant |\mathcal{D}_V| \leqslant \Delta_\Gamma$ since CG are node-transitive [28, Theorem 3.7].

Lemma 7.4.2 *Algorithm 9 computes the shortest node-disjoint paths between any two nodes of a* CG *in time* $O(K|\mathbb{D}|\Delta_\Gamma l^3)$, *where the longest path computed has length* l *and is the* K*-th shortest path.*

Proof. This proof is analogous to the proof of Lemma 7.4.1. ∎

Algorithm 9 Compute the shortest node-disjoint paths.

Input: Label of the source node $u \in L$.
Input: Label of the destination node $v \in L$.
Input: Value of the network degree Δ_Γ.
Input: Value of the network diameter D_Γ.
Input: Value of the *fellow-traveler* constant k.
Output: A set of words $\mathcal{D}_V(u, v) \subset \mathcal{F}(\mathcal{A})$ representing the shortest node-disjoint paths from $\overline{u}$ to $\overline{v}$.

1: $i \leftarrow 1$, $[w]_{old} \leftarrow \emptyset$, $\mathcal{D}_V \leftarrow \emptyset$, $V_D \leftarrow \emptyset$.
2: Compute the minimal paths $\mathbb{P}_{\lceil D_\Gamma/k \rceil}(w)$ (Algorithm 5).
3: $[w]_{new} \leftarrow \mathbb{P}_{\lceil D_\Gamma/k \rceil}(w)$.
4: **while** $([w]_{new} \neq [w]_{old})$ **do**
5: **for** $r \in [w]_{new} \setminus [w]_{old}$ **do**
6: Compute $V_{\widehat{r}}$ (Lemma 7.1.3).
7: **if** $V_D \cap V_{\widehat{r}} = \emptyset$ **then**
8: Add r to $\mathcal{D}_V$.
9: $V_D \leftarrow V_D \cup V_{\widehat{r}}$.
10: **if** $|\mathcal{D}_V| = \Delta_\Gamma$ **then**
11: **return** $\mathcal{D}_V$.
12: Compute $[w]_i$ (Algorithm 6).
13: $[w]_{old} \leftarrow [w]_{new}$, $[w]_{new} \leftarrow [w]_i$, $i \leftarrow i + 1$.
14: **return** $\mathcal{D}_V$.

7.5 COMPUTING THE SHORTEST PATHS AVOIDING A SET OF LINKS AND NODES

Let $E_f \subset L \times L$ represent a set of links in $E(\Gamma)$ and let $V_f \subset L$ represent a set of nodes in $V(\Gamma)$. Given E_f, V_f and the labels of two nodes $u, v \in L \setminus V_f$, Algorithm 10 computes the shortest path from $\overline{u}$ to $\overline{v}$ containing neither link in E_f nor nodes in V_f, if it exists. Otherwise, it returns *Null*. Similarly to Algorithm 8 and Algorithm 9, the set of the minimal paths $\mathbb{P}_{\lceil D_\Gamma/k \rceil}$ is computed (line 2); and then the set of paths $[w]_i$ is sequentially computed for $i \geqslant 0$ (lines 12-13). For each $r \in [w]_i \setminus [w]_{i-1}$, the links and intermediate nodes in $\widehat{r}(\overline{u})$ are computed (lines 6-7). If $\widehat{r}(\overline{u})$ does not contain neither links in E_f nor nodes in V_f, then r is returned (*if* condition at line 8). Otherwise, the process is repeated for $i + 1$, and so on until the path is found, if it exists (*while* loop at line 4).

Lemma 7.5.1 *Algorithm 10 computes the shortest paths between any two nodes of a CG, such that the computed paths avoid a set of links and nodes. The algorithm runs in time* $O(k|\mathbb{D}|\Delta_\Gamma l^3)$*, where the longest path computed has length* l *and is the* k*-th shortest path.*

Proof. This proof is analogous to the proof of Lemma 7.4.1. ■

Example 7.5.1 *Consider the set of tuples* $E_f = \{(aba, ba), (b, bc)\}$*, the set of words* $V_f = \{bac, b\}$*, and the nodes* $\overline{aba}$*,* $\overline{bc}$ *in* $BS(4)$*. To compute the shortest path from* $\overline{bac}$ *to* $\overline{b}$ *that avoids avoiding links in* E_f *and nodes in* V_f*, Algorithm 10 proceeds as follows:*

1) $i \leftarrow 1$, $[w]_{old} \leftarrow \emptyset$.

Algorithm 10 Compute the shortest paths avoiding a set of links and nodes.

Input: Label of the source node $u \in L$.
Input: Label of the destination node $v \in L$.
Input: A set of tuples $E_f \subset L \times L$ representing a set of links in $E(\Gamma)$.
Input: A set of words $V_f \subset L$ representing a set of nodes in $V(\Gamma)$, such that $u, v \notin V_f$.
Input: Value of the network diameter D_Γ.
Input: Value of the *fellow-traveler* constant k.
Output: A word $r \in \mathcal{F}(\mathcal{A})$ representing the shortest path from $\overline{u}$ and $\overline{v}$ containing neither links in E_f nor nodes in V_f. *Null*, if such a path does not exist.

1: $i \leftarrow 1$, $[w]_{old} \leftarrow \emptyset$.
2: Compute the minimal paths $\mathbb{P}_{\lceil D_\Gamma/k \rceil}(w)$ (Algorithm 5).
3: $[w]_{new} \leftarrow \mathbb{P}_{\lceil D_\Gamma/k \rceil}(w)$.
4: **while** $([w]_{new} \neq [w]_{old})$ **do**
5: **for** $r \in [w]_{new} \setminus [w]_{old}$ **do**
6: Compute $E_{\widehat{r}}(\overline{u})$ (Lemma 7.1.2).
7: Compute $V_{\widehat{r}}(\overline{u})$ (Lemma 7.1.3).
8: **if** $(E_f \cap E_{\widehat{r}}(\overline{u}) = \emptyset)$ AND $(V_f \cap \mathcal{V}_{\widehat{r}}(\overline{u}) = \emptyset)$ **then**
9: **return** r.
10: Compute $[w]_i$ (Algorithm 6).
11: $[w]_{old} \leftarrow [w]_{new}$, $[w]_{new} \leftarrow [w]_i$, $i \leftarrow i + 1$.
12: **return** *Null*.

2) $\mathbb{P}_3(w) \leftarrow \{bac, bca\}$ *(in* $BS(4)$, $k = 2$ *and* $D_\Gamma = 6$*).*

3) $[w]_{new} \leftarrow \{bac, bca\}$.

4) $[w]_{new} = \{a\}$ *and* $[w]_{old} = \emptyset$*, then*

a) $r \leftarrow bac$

i. $E_{\widehat{bac}}(\overline{aba}) \leftarrow [(aba, ba), (ba, b), (b, bc)]$.

ii. $V_{\widehat{bac}}(\overline{aba}) \leftarrow [(ba, b]$.

iii. $E_f \cap E_{\widehat{bac}}(\overline{aba}) = \{(aba, ba), (b, bc)\}$ *and* $V_f \cap V_{\widehat{bac}}(\overline{aba}) = \{b\}$.

b) $r \leftarrow bca$.

i. $E_{\widehat{bca}}(\overline{aba}) \leftarrow [(aba, ba), (ba, bac), (bac, bc)]$.

ii. $V_{\widehat{bca}}(\overline{aba}) \leftarrow [(ba, bac]$.

iii. $E_f \cap E_{\widehat{bca}}(\overline{aba}) = \{(aba, ba)\}$ *and* $V_f \cap V_{\widehat{bca}}(\overline{aba}) = \{bac\}$.

c) $[w]_1 \leftarrow \{bac, bca, ababc, cbcba\}$.

d) $[w]_{old} \leftarrow \{bac, bca\}$, $[w]_{new} \leftarrow \{bac, bca, ababc, cbcba\}$, $i \leftarrow 2$

e) $r \leftarrow ababc$.

i. $E_{\widehat{ababc}}(\overline{aba}) \leftarrow \{(aba, ab), (ab, a), (a, e_{\mathcal{A}}), (e_{\mathcal{A}}, b), (b, bc)\}$.

ii. $V_{\widehat{ababc}}(\overline{aba}) \leftarrow \{ab, a, e_{\mathcal{A}}, b\}$.

iii. $E_f \cap E_{\widehat{ababc}}(\overline{aba}) = \{(b, bc)\}$ *and* $V_f \cap V_{\widehat{ababc}}(\overline{aba}) = \{b\}$.

f) $r \leftarrow cbcba$.

i. $E_{\widehat{cbcba}}(\overline{aba}) \leftarrow \{(aba, abac), (abac, abacb), (abacb, bacb), (bacb, bac), (bac, bc)\}$.

ii. $V_{\widehat{cbcba}}(\overline{aba}) \leftarrow \{abac, abacb, bacbbac\}$.

iii. $E_f \cap E_{\widehat{cbcba}}(\overline{aba}) = \emptyset$ *and* $V_f \cap V_{\widehat{cbcba}}(\overline{aba}) = \emptyset$, *then*

- *Return* $r = cbcba$.

Therefore, the shortest path from $\overline{aba}$ *to* $\overline{bc}$ *that avoids links in* E_f *and paths in* V_f *is* $\widehat{cbcba}$.

8

FAULT-TOLERANT MECHANISM

The WPR supports multiple failures of both nodes and links. In addition, it provides minimal routing paths in spite of nodes do not keep a global record of failures. This chapter presents the fault-tolerant mechanism of the WPR. First, the general operation of this mechanism is described. Then, it is explained how nodes keep and update its local records of failures. Finally, the processes of notification of failures and recovery from failures, respectively, are presented.

8.1 GENERAL OPERATION

To provide fault-tolerance, every node in the network keeps partial records of faulty nodes and links. Then nodes use Algorithm 10 to compute the routing paths avoiding the recorded failures. Since the computed paths could contains failures that were not recorded, the fault-tolerant mechanism applies hop-by-hop routing in mode single-path. Recall from Section 1.2.2.3 that in hop-by-hop routing, the routing path is computed by each node in the path. Then, it does not matter that the computed path contains failures, provided that the next node in such a path is correct. Table 16 introduces the notation used for defining the fault-tolerant mechanism. Recall from Table 14 that the network topology is defined by $\Gamma(\mathcal{G}, S)$.

Table 16: Notation of the fault-tolerant mechanism.

Parameter	Definition
$V_F \subset L$	Labels of all faulty nodes in the network.
$E_F \subset L \times L$	Labels of all faulty links in the network.
$\mathcal{V}_u \subseteq V_F$	Faulty nodes recorded at node $\overline{u}$.
$\mathcal{E}_u \subseteq E_F$	Faulty links recorded at node $\overline{u}$.
$f \in V_F$	Label of a faulty node.
$(e, f) \in E_F$	Label of a faulty link.
$r \in V_\Gamma$	Label of a recovered node.
$(r, t) \in E_\Gamma$	Label of a recovered link.

8.2 FAILURES' RECORDS

As it is shown in Table 16, every node $\overline{u}$ in the network keeps partial records of faulty nodes and links, which are denoted by $\mathcal{V}_u$ and $\mathcal{E}_u$, respectively. This section focuses on the criterion to determine whether a node $\overline{u}$ notified of a failure must update $\mathcal{V}_u$ and/or $\mathcal{E}_u$. It is important to mention that updating a record of failures refers to adding or remov-

ing elements from it. The process of failure notification is presented in Section 8.3.

8.2.1 *Updating the faulty nodes record*

Algorithm 11 Update the faulty nodes record $\mathcal{V}_u$ with the faulty node $\bar{f}$.

Input: Label of node $u \in L$.
Input: Label of the faulty node $f \in L$.
Input: Faulty nodes record $\mathcal{V}_u \subset L$.
Input: Faulty links record $\mathcal{E}_u \subset L \times L$.
Output: *True*, if $\mathcal{V}_u$ is updated. Otherwise, *False*.

1: $update \leftarrow$ *False*.
2: **for** $(d, e) \in \mathcal{E}_u$ **do**
3: **if** $f \in (d, e)$ **then**
4: Remove (d, e) from $\mathcal{E}_u$.
5: $update \leftarrow$ *True*.
6: **if** $update$ **then**
7: Add f to $\mathcal{V}_u$.
8: **return** *True*.
9: **for** $v \in \{f \cdot x : x \in A\} \setminus \{\mathcal{V}_u \cup \{f\}\}$ **do**
10: $w_1 \leftarrow$ the *shortLex* path from $\bar{u}$ to $\bar{v}$ avoiding nodes in $\mathcal{V}_u \cup \{f\}$ and links in $\mathcal{E}_u$ (Algorithm 10).
11: $w_2 \leftarrow$ the *shortLex* path from $\bar{u}$ to $\bar{v}$ avoiding nodes in $\mathcal{V}_u \setminus \{f\}$ and links in $\mathcal{E}_u$ (Algorithm 10).
12: **if** ($w_1 = Null$ AND $f \notin \mathcal{V}_u$) OR ($w_1(1) \neq w_2(1)$ AND $f \notin \mathcal{V}_u$) **then**
13: Add f to $\mathcal{V}_u$.
14: **return** *True*.
15: **if** $f \in \mathcal{V}_u$ **then**
16: Remove f from $\mathcal{V}_u$.
17: **return** *True*.
18: **return** *False*.

Algorithm 11 presents the steps to determine whether a node $\bar{u}$ notified about the failure of a node $\bar{f}$ must update $\mathcal{V}_u$. If $\bar{f}$ is recorded in $\mathcal{V}_u$, then it is not necessary to record its incident links in $\mathcal{E}_u$. The algorithm returns *True* if $\mathcal{V}_u$ is updated. Otherwise, it returns *False*. First, links in $\mathcal{E}_u$ that are adjacent to $\bar{f}$ are searched and removed from $\mathcal{E}_u$ (*for* loop at line 2). If some element was removed from $\mathcal{E}_u$, then f is added to $\mathcal{V}_u$ and the algorithm returns *True* (*if* condition at line 6). Otherwise, the algorithm proceeds as follows. For each node $\bar{v}$ adjacent to $\bar{f}$ (*for* loop at line 9), the following paths from $\bar{u}$ to $\bar{v}$ are computed:

- The *shortLex* path avoiding failures in $\mathcal{V}_u \cup \{f\}$ and $\mathcal{E}_u$, said path is denoted by $\widehat{w}_1$ (line 10).
- The *shortLex* path avoiding failures in $\mathcal{V}_u$ and $\mathcal{E}_u$, said path is denoted by $\widehat{w}_2$ (line 11).

The label f is recorded in $\mathcal{V}_u$ if $f \notin \mathcal{V}_u$ and for some $\bar{v}$ (*if* condition at line 12):

1) it is not possible to compute $\widehat{w}_1$, i.e. $w_1 = \mathit{Null}$, or

2) paths $\widehat{w}_1$ and $\widehat{w}_2$ begin at different links, i.e. $w_1(1) \neq w_2(1)$.

Conversely, if $f \in \mathcal{V}_u$ and for every $\bar{v}$, the paths $\widehat{w}_1$ and $\widehat{w}_2$ begin at the same link (*if* condition at line 15), then the next nodes in the computed paths are the same whether f is recorded or not. Therefore, the label f is removed from $\mathcal{V}_u$. The algorithm returns *True* whether f has been added in or removed from $\mathcal{V}_u$. Otherwise, it returns *False*. The following corollary results from this process:

Corollary 8.2.1 *Upon failing a node $\bar{f}$, every node $\bar{u}$ will add f to $\mathcal{V}_u$ if: 1) after node $\bar{f}$ fails, the network is disconnected, or 2) before node $\bar{f}$ fails, $\bar{u}$ and $\bar{f}$ were in a chordless cycle.*

Lemma 8.2.1 *Let $\bar{u}$ be a node notified about the failure of a node $\bar{f}$. Let $\widehat{w}$ be the longest path among the shortest paths (avoiding failures) from $\bar{u}$ to the adjacent nodes to $\bar{f}$. Let l denote the length of $\widehat{w}$ and let K be an integer, such that $\widehat{w}$ is the K-th shortest path in the network without failures, i.e. $\Gamma(\mathcal{G}, S)$, from $\bar{u}$ to $\bar{f}$. Algorithm 11 updates the record of node failures of $\bar{u}$ in time $O(K|\mathbb{D}|\Delta_\Gamma^2 l^3)$.*

Proof. The paths avoiding failures from $\bar{u}$ to each node adjacent to $\bar{f}$ are computed. Each path is computed in time $O(K|\mathbb{D}|\Delta_\Gamma l^3)$ due to Lemma 7.5.1. Therefore Algorithm 11 runs in time $O(K|\mathbb{D}|\Delta_\Gamma^2 l^3)$. ■

8.2.2 *Updating the faulty links record*

Algorithm 12 determines whether a node $\bar{u}$ notified about the failure of a link (e, f) must update $\mathcal{E}_u$. Similarly to Algorithm 11, for each node $\bar{v}$ incident to (e, f) (*for* loop at line 1), the following paths are computed:

- The *shortLex* path from $\bar{u}$ to $\bar{v}$ avoiding the recorded failures and the faulty link (e, f), i.e. $\widehat{w}_1$ (line 2).
- The *shortLex* path from $\bar{u}$ to $\bar{v}$ avoiding the recorded failures except the faulty link (e, f), i.e. $\widehat{w_2}$ (line 3).

The tuple (e, f) is recorded in $\mathcal{E}_u$ if $(e, f) \notin \mathcal{E}_u$ and for some $\bar{v}$ (*if* condition at line 4):

1) it is not possible to compute $\widehat{w}_1$, i.e. $w_1 = \mathit{Null}$, or

2) paths $\widehat{w}_1$ and $\widehat{w}_2$ begin at different links, i.e. $w_1(1) \neq w_2(1)$.

Conversely, if $(e, f) \in \mathcal{E}_u$ and for every $\bar{v}$, paths $\widehat{w}_1$ and $\widehat{w}_2$ begin at the same link (*if* condition at line 7), then the next nodes in the computed paths are the same whether f is recorded or not. Therefore, the tuple (e, f) is removed from $\mathcal{E}_u$. The algorithm returns *True* whether (e, f) has been added in or removed from $\mathcal{E}_u$. Otherwise, it returns *False*. The following corollary results from this process:

Corollary 8.2.2 *Upon failing a link (e, f), every node $\bar{u}$ will add (e, f) to $\mathcal{E}_u$ if: 1) after link (e, f) fails, the network is disconnected, or 2) before link (e, f) fails, $\bar{u}$ and (e, f) were in a chordless cycle.*

Algorithm 12 Update the faulty links record $\mathcal{E}_u$ with the faulty link (e, f).

Input: Label of node $u \in L$.
Input: Label of a faulty link $(e, f) \in L \times L$.
Input: Faulty nodes record $\mathcal{V}_u \subset L$.
Input: Faulty links record $\mathcal{E}_u \subset L \times L$.
Output: *True*, if $\mathcal{E}_u$ is updated. Otherwise, *False*.

1: **for** $v \in \{e, f\} \setminus \mathcal{V}_u$ **do**
2: $w_1 \leftarrow$ the *shortLex* path from $\overline{u}$ to $\overline{v}$ avoiding nodes in $\mathcal{V}_u$ and links in $\mathcal{E}_u \cup \{(e, f)\}$ (Algorithm 10).
3: $w_2 \leftarrow$ the *shortLex* path from $\overline{u}$ to $\overline{v}$ avoiding nodes in $\mathcal{V}_u$ and links in $\mathcal{E}_u \setminus \{(e, f)\}$ (Algorithm 10).
4: **if** ($w_1 = Null$ AND $(e, f) \notin \mathcal{E}_u$) OR ($w_1(1) \neq w_2(1)$ AND $(e, f) \notin \mathcal{E}_u$) **then**
5: Add (e, f) to $\mathcal{E}_u$.
6: **return** *True*.
7: **if** $(e, f) \in \mathcal{E}_u$ **then**
8: Remove (e, f) from $\mathcal{E}_u$.
9: **return** *True*.
10: **return** *False*.

Lemma 8.2.2 *Let $\overline{u}$ be a node notified about the failure of a link (e, f). Let $\widehat{w}$ be the longest path among the shortest paths (avoiding failures) from $\overline{u}$ to each node in $\{\overline{e}, \overline{f}\}$. Let l denote the length of $\widehat{w}$ and let K be an integer, such that $\widehat{w}$ is the K-th shortest path in the network (without failures), i.e. $\Gamma(\mathcal{G}, S)$. Algorithm 12 updates the record of link failures of $\overline{u}$ in time $O(K|\mathbb{D}|\Delta_\Gamma l^3)$.*

Proof. The paths avoiding failures from $\overline{u}$ to each node in $\{e, f\}$ are computed. Each path is computed in time $O(K|\mathbb{D}|\Delta_\Gamma l^3)$ due to Lemma 7.5.1. Therefore Algorithm 12 runs in time $O(K|\mathbb{D}|\Delta_\Gamma l^3)$. ■

8.3 NOTIFICATIONS OF FAULTY NODES AND LINKS

The notification processes of faulty nodes and links are presented in Algorithm 13 and Algorithm 14, respectively. When a node or link fails, the failure notification beginss at each node $\overline{z}$ adjacent to the faulty node or incident to the faulty link. This process proceeds as follows. First, the label of the faulty node or link is computed by concatenating the label of node $\overline{z}$ and the port connected to the faulty element. Then, the resulting label is added to the corresponding record of failures. Finally, whether the failure corresponds to a node or link, a message `FAULTY_NODE` or `FAULTY_LINK` is sent through the active ports of $\overline{z}$. This message contains the label of the faulty element and the failure records of $\overline{z}$.

Each node $\overline{u}$ receiving a failure notification message (from a node $\overline{v}$) updates its record of failures (if it is necessary) executing whether Algorithm 11 or Algorithm 12 according to the kind of message. If the failures records are not updated, node $\overline{u}$ ends its process of failure notification. Otherwise, for each failure recorded in $\overline{v}$ but not in $\overline{u}$, the corresponding faulty record is updated. Note that recording a new failure could involve recording some failures in $\mathcal{V}_v$. Finally, a new message of failure notifica-

tion is sent through the active ports of $\bar{u}$ except for the one connected to $\bar{v}$. Figure 18 illustrates the process of node failure notification in $BS(4)$.

Algorithm 13 Notification of a faulty node.

Upon failing a node $\bar{f}$, each node $\bar{z}$ adjacent to $\bar{f}$ **do**:

1: $f \leftarrow$ the canonical form of $z \cdot x$, where x is the port connected to the faulty node $\bar{f}$.
2: Add f to $\mathcal{V}_z$.
3: Send out the message $\texttt{FAULTY\_NODE}(f, \mathcal{V}_z, \mathcal{E}_z)$ through its active ports.

Every node $\bar{u}$ receiving a message $\texttt{FAULTY\_NODE}(f, \mathcal{V}_v, \mathcal{E}_v)$ through its port x **do**:

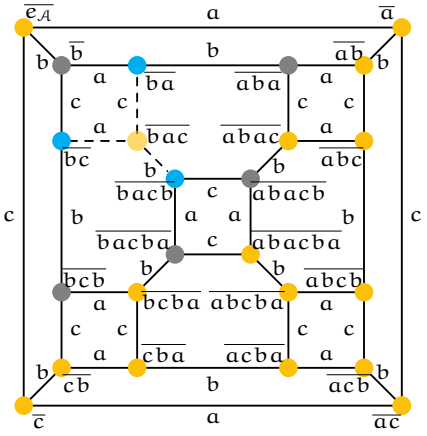
1: $\texttt{update} \leftarrow$ update $\mathcal{V}_u$ with f (Algorithm 11).
2: **if** $\texttt{update}$ **then**
3: **for** $f \in \mathcal{V}_v \setminus \mathcal{V}_u$ **do**
4: Update $\mathcal{V}_u$ with f (Algorithm 11).
5: **for** $(e, f) \in \mathcal{E}_v \setminus \mathcal{E}_u$ **do**
6: Update $\mathcal{E}_u$ with (e, f) (Algorithm 12).
7: Send out a message $\texttt{FAULTY\_NODE}(f, \mathcal{V}_u, \mathcal{E}_u)$ through its active ports except x.

Lemma 8.3.1 *Let $\bar{f}$ be a new faulty node, such that the largest chordless that includes $\bar{f}$ (after node $\bar{f}$ fails) has length c. Then the notification of the faulty node $\bar{f}$ takes $O(\max\{1, |V_\Gamma|, |E_\Gamma|\}K|\mathbb{D}|\Delta_\Gamma^2 \lceil \frac{c}{2} \rceil^4)$ time units and requires $O(\Delta_\Gamma |N(\bar{f}, \lceil \frac{c}{2} \rceil)|)$ messages applying Algorithm 13.*

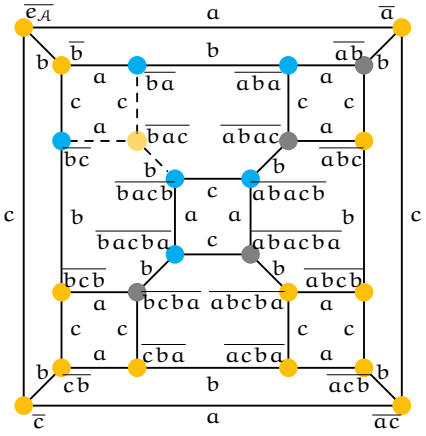
Proof. Regarding to the time complexity. When a node $\bar{u}$ at distance l from node $\bar{f}$ receives a message $\texttt{NODE\_FAILURE}$ from a node $\bar{v}$, it tries to update its record of node failures on the new failure, i.e. $\bar{f}$, which takes $O(K|\mathbb{D}|\Delta_\Gamma^2 l^3)$ due to Lemma 8.2.1. If the record of node failures was updated, then it is again updated on the new node and link failures recorded by $\bar{v}$, i.e $|\mathcal{V}_v \setminus \mathcal{V}_u| + |\mathcal{E}_v \setminus \mathcal{E}_u|$ times. In the worst case, $|\mathcal{V}_v \setminus \mathcal{V}_u| + |\mathcal{E}_v \setminus \mathcal{E}_u| = |V_\Gamma| + |E_\Gamma|$, and hence $\bar{u}$ updates its records of failures in time $O(\max\{1, |V_\Gamma|, |E_\Gamma|\}K|\mathbb{D}|\Delta_\Gamma^2 l^3)$ due to Lemma 8.2.1 and Lemma 8.2.2. By Corollary 8.2.1, the last notified node is at distance $\lceil \frac{c}{2} \rceil$ from $\bar{f}$. Then assuming that each message $\texttt{NODE\_FAILURE}$ incurs in a delay of at most one time unit, the process of notification of a faulty node takes
$1 + \sum_{l=1}^{\lceil \frac{c}{2} \rceil} (\max\{1, |V_\Gamma|, |E_\Gamma|\}K|\mathbb{D}|\Delta_\Gamma^2 \lceil \frac{c}{2} \rceil^3) \approx O(\max\{1, |V_\Gamma|, |E_\Gamma|\}K|\mathbb{D}|\Delta_\Gamma^2 \lceil \frac{c}{2} \rceil^4)$
time units from the beginning of the execution. Turning now to the message complexity. All notified nodes are in the $\lceil \frac{c}{2} \rceil$-neighborhood of $\bar{f}$, i.e. $N(\bar{f}, \lceil \frac{c}{2} \rceil)$, see Definition 2.1.11, due to Corollary 8.2.1. Since each notified node sends at most $\Delta_\Gamma - 1$ messages, the total number of messages transmitted during the notification of a faulty node is $O(\Delta_\Gamma |N(\bar{f}, \lceil \frac{c}{2} \rceil)|)$. ■

Lemma 8.3.2 *Let (e, f) be a new faulty link, such that largest chordless that includes (e, f) (after link (e, f) fails) has length c. Then the notification of the faulty link (e, f) takes $O(\max\{1, |V_\Gamma|, |E_\Gamma|\}K|\mathbb{D}|\Delta_\Gamma^2 \lceil \frac{c}{2} \rceil^4)$ time units and requires $O(\Delta_\Gamma |N(\bar{f}, \lceil \frac{c}{2} \rceil)|)$ messages applying Algorithm 14.*

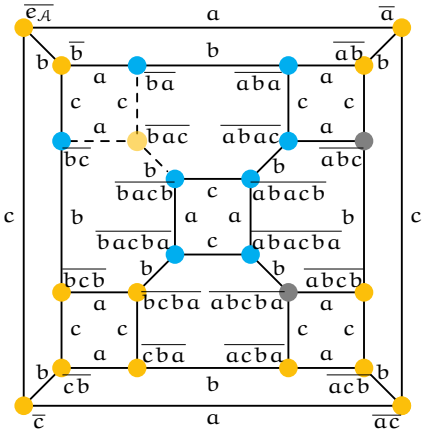
Proof. This proof proceeds as the proof of Lemma 8.3.1. ■



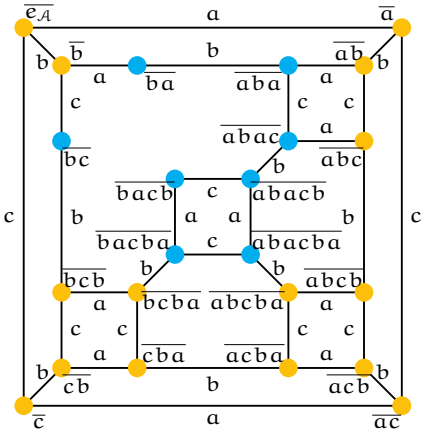
(a) Node bac fails and their adjacent nodes ba, bc and bacb add bac to their failures records, and then notify their adjacent nodes (gray nodes).



(b) Nodes aba, abac and bacba add bac to their failures records, and then notify their adjacent nodes (gray nodes).



(c) Nodes abac and abacba add bac to their failures records, and then notify their adjacent nodes (gray nodes).



(d) Final state of the network. The blue nodes have updated their failures records.

Notified nodes	**Nodes adjacent to** $\overline{bac}$							
	$\overline{ba}$		$\overline{bc}$		$\overline{bacba}$			
$(\overline{u})$	$\widehat{w_1}$	$\widehat{w_2}$	$\widehat{w_1}$	$\widehat{w_2}$	$\widehat{w_1}$	$\widehat{w_2}$	$\mathcal{V}'_u$	$\mathcal{V}_u$
$\overline{bcb}$	$\widehat{bca}$	$\widehat{bac}$	$\widehat{b}$	$\widehat{b}$	$\widehat{aba}$	$\widehat{aba}$	$\emptyset$	$\emptyset$
$\overline{b}$	$\widehat{a}$	$\widehat{a}$	$\widehat{c}$	$\widehat{c}$	$\widehat{abcbc}$	$\widehat{acb}$	$\emptyset$	$\emptyset$
$\overline{bacba}$	$\widehat{babca}$	$\widehat{abc}$	$\widehat{bab}$	$\widehat{aba}$	$\widehat{a}$	$\widehat{a}$	$\emptyset$	$\{bac\}$
$\overline{abacb}$	$\widehat{bcb}$	$\widehat{bcb}$	$\widehat{cba}$	$\widehat{bcbac}$	$\widehat{c}$	$\widehat{c}$	$\emptyset$	$\{bac\}$
$\overline{aba}$	$\widehat{b}$	$\widehat{b}$	$\widehat{bac}$	$\widehat{bac}$	$\widehat{bcb}$	$\widehat{cbc}$	$\emptyset$	$\emptyset$
$\overline{ab}$	$\widehat{ab}$	$\widehat{ab}$	$\widehat{abac}$	$\widehat{abac}$	$\widehat{acbc}$	$\widehat{abcb}$	$\emptyset$	$\emptyset$
$\overline{abac}$	$\widehat{cb}$	$\widehat{cb}$	$\widehat{cbac}$	$\widehat{bcba}$	$\widehat{bc}$	$\widehat{bc}$	$\emptyset$	$\{bac\}$
$\overline{abacba}$	$\widehat{abcb}$	$\widehat{abcb}$	$\widehat{cbab}$	$\widehat{acba}$	$\widehat{ac}$	$\widehat{ac}$	$\emptyset$	$\{bac\}$
$\overline{bcba}$	$\widehat{abca}$	$\widehat{abac}$	$\widehat{ab}$	$\widehat{ab}$	$\widehat{bc}$	$\widehat{bc}$	$\emptyset$	$\emptyset$
$\overline{abc}$	$\widehat{acb}$	$\widehat{acb}$	$\widehat{abcba}$	$\widehat{acbac}$	$\widehat{abc}$	$\widehat{abc}$	$\emptyset$	$\emptyset$
$\overline{abcba}$	$\widehat{abacb}$	$\widehat{abacb}$	$\widehat{bcbab}$	$\widehat{bacba}$	$\widehat{bac}$	$\widehat{bac}$	$\emptyset$	$\emptyset$

(e) Paths computed by notified nodes. If paths begin at different letters, the failure is recorded. Sets $\mathcal{V}'_u$ and $\mathcal{V}_u$ denote the record after and before notification, respectively.

Figure 18: Process of node failure notification in BS(4).

Algorithm 14 Notification of a faulty link.

Upon failing a link (e,f), each node $\bar{z}$ incident to (e,f) **do**:

1: $f \leftarrow$ the canonical form of $z \cdot x$, where x is the port connected to the faulty link (e, f).
2: Add (e, f) to $\mathcal{E}_z$, where $e = z$.
3: Send out the message `FAULTY_LINK`$((e, f), \mathcal{V}_z, \mathcal{E}_z)$ through its active ports.

Every node $\bar{u}$ receiving a message `FAULTY_LINK`$((e, f), \mathcal{V}_v, \mathcal{E}_v)$ through its port x **do**:

1: update $\leftarrow$ update $\mathcal{E}_u$ with (v, f) (Algorithm 12).
2: **if** update **then**
3: **for** $f \in \mathcal{V}_v \setminus \mathcal{V}_u$ **do**
4: Update $\mathcal{V}_u$ with f (Algorithm 11).
5: **for** $(v, f) \in \mathcal{E}_v \setminus \mathcal{E}_u$ **do**
6: Update $\mathcal{E}_u$ with (e, f) (Algorithm 12).
7: Send out a message `FAULTY_LINK`$((e, f), \mathcal{V}_u, \mathcal{E}_u)$ through its active ports except x.

8.4 NOTIFICATION OF RECOVERED NODES AND LINKS

The notification processes of nodes and links recovered from a failure are presented in Algorithm 15 and Algorithm 16. Similarly to a failure notification, a recovery notification is initiated in each node $\bar{z}$ adjacent to the recovered node or incident to the recovered link. First, the label of the recovered element is computed and removed from the corresponding record of failures. Then, for each failure recorded in $\bar{u}$, the corresponding faulty record is updated, which could involve remove other elements from the records of failures. Finally, whether the recovered element is a node or a link, a message `RECOVERED_NODE` or `RECOVERED_LINK` is sent through the active ports of $\bar{z}$ except for the one connected to the recovered element. This message contains the label of the recovered element.

Each node $\bar{u}$ receiving a recovery notification message (from a node $\bar{v}$) checks if the recovered element is in the record of failures. If not, node $\bar{u}$ ends the process of recovery notification. Otherwise, the label of the recovered element is removed from the corresponding record of failures. Then, for each failure recorded in $\bar{u}$, the records of failures are updated. Finally, a new message of recovery notification is sent through the active ports of $\bar{u}$ except for the one connected to $\bar{v}$.

Lemma 8.4.1 *Let $\bar{r}$ be a new recovered node, such that largest chordless that includes $\bar{r}$ (after recovering node $\bar{r}$) has length c. Then the notification of the recovered node $\bar{r}$ takes $O(\max\{1, |V_\Gamma|, |E_\Gamma|\}K|\mathbb{D}|\Delta_\Gamma^2 \lceil \frac{c}{2} \rceil^4)$ time units and requires $O(\Delta_\Gamma |N(\bar{f}, \lceil \frac{c}{2} \rceil)|)$ messages applying Algorithm 15.*

Proof. This proof proceeds as the proof of Lemma 8.3.1. ■

Lemma 8.4.2 *Let (r, t) be a new recovered link, such that largest chordless that includes (r, t) (after recovering link (r, t)) has length c. Then the notification of the recovered link (r, t) takes $O(\max\{1, |V_\Gamma|, |E_\Gamma|\}K|\mathbb{D}|\Delta_\Gamma^2 \lceil \frac{c}{2} \rceil^4)$ time units and requires $O(\Delta_\Gamma |N(\bar{f}, \lceil \frac{c}{2} \rceil)|)$ messages applying Algorithm 16.*

Algorithm 15 Notification of a recovered node.

Upon recovering a node $\overline{r}$ after failing, each node $\overline{z}$ adjacent to $\overline{r}$ **do**:

1: $r \leftarrow$ the canonical form of $z \cdot x$, where x is the port connected to the recovered node $\overline{r}$.
2: Remove r from $\mathcal{V}_z$.
3: **for** $f \in \mathcal{V}_z$ **do**
4: Update $\mathcal{V}_z$ with f (Algorithm 11).
5: **for** $(e, f) \in \mathcal{E}_z$ **do**
6: Update $\mathcal{E}_z$ with (e, f) (Algorithm 12).
7: Send out the message `RECOVERED_NODE`(r) through its active ports except x.

Every node $\overline{u}$ receiving a message `RECOVERED_NODE`(r) through its port x **do**:

1: **if** $r \in \mathcal{V}_u$ **then**
2: Remove r from $\mathcal{V}_u$.
3: **for** $f \in \mathcal{V}_u$ **do**
4: Update $\mathcal{V}_u$ with f (Algorithm 11).
5: **for** $(e, f) \in \mathcal{E}_u$ **do**
6: Update $\mathcal{E}_u$ with (e, f) (Algorithm 12).
7: Send out the message `RECOVERED_NODE`(r) through its active ports except x.

Algorithm 16 Notification of a recovered link.

Upon recovering a link (r, t) after failing, each node $\overline{z}$ incident to (r, t) **do**:

1: $t \leftarrow$ the canonical form of $z \cdot x$, where x is the port connected to the recovered link (r, t).
2: Remove (r, t) from $\mathcal{E}_z$, where $r = z$.
3: **for** $f \in \mathcal{V}_z$ **do**
4: Update $\mathcal{V}_z$ with f (Algorithm 11).
5: **for** $(e, f) \in \mathcal{E}_z$ **do**
6: Update $\mathcal{E}_z$ with (e, f) (Algorithm 12).
7: Send out the message `RECOVERED_LINK`(r, t) through its active ports except x.

Every node $\overline{u}$ receiving a message `RECOVERED_LINK`(r, t) through its port x **do**:

1: **if** $(r, t) \in \mathcal{E}_u$ **then**
2: Remove (r, t) from $\mathcal{V}_u$.
3: **for** $f \in \mathcal{E}_z$ **do**
4: Update $\mathcal{V}_z$ with f (Algorithm 11).
5: **for** $(e, f) \in \mathcal{E}_z$ **do**
6: Update $\mathcal{E}_z$ with (e, f) (Algorithm 12).
7: Send out the message `RECOVERED_LINK`(r, t) through its active ports except x.

Proof. This proof proceeds as the proof of Lemma 8.3.1. ■

Everything flows.

— Heraclitus

9 FORWARDING PROTOCOLS

The forwarding protocols operate according to the kind of routing that is working. Further details about the kinds of routing and their general operation are presented in Section 1.2.2.3. This chapter presents the forwarding protocols for: 1) deterministic routing, which applies source routing in modes single-path and multi-path; and 2) fault-tolerant routing, which applies hop-by-hop routing in mode single-path.

9.1 DETERMINISTIC ROUTING

In the WPR, each node is able to compute the shortest paths for a given destination, such paths are represented by words denoting a sequence of ports, see Section 4.3. Thereby, deterministic routing applies source routing, where the routing paths are computed only once for each message. When a source node computes a routing path $w = x_1 x_2 \dots x_l$, the first letter in the path corresponds to the output port. The remaining substring denotes the rest of the path, which is used as the new message header.

Lemma 9.1.1 *A message header defined by the WPR, in deterministic mode, can be represented with* $O(D_\Gamma \log(\Delta_\Gamma))$ *bits.*

Proof. A header in deterministic mode is given by the word representing the *ShortLex* path from the next node in the routing path to the destination node, thereby this proof is analogous to the proof of Lemma 6.1.1. ■

Algorithm 17 and Algorithm 18 present the steps for single-path and multi-path forwarding in source nodes. Note that when the next node in the path is the destination node, i.e. $l = 1$, the header consists of just the symbol \$ representing the empty path. as it is shown in Figure 14, Algorithm 7, Algorithm 8 and Algorithm 9 provide the necessary path diversity for multi-path. Depending on the routing requirements, any of them can be used in the step 1 of Algorithm 18. Finally, Algorithm 19 presents the steps for forwarding in intermediate nodes, which is the same for both modes single-path and multi-path.

Lemma 9.1.2 *The forwarding decision in deterministic single-path mode of the WPR is taken by source nodes using Algorithm 17, which runs in time* $O(|\mathbb{D}|\Delta_\Gamma D_\Gamma^2)$ *due to Lemma 7.2.1.*

Lemma 9.1.3 *The forwarding decision in deterministic multi-path mode of the WPR is taken by source nodes using Algorithm 18, which runs in time* $O(K|\mathbb{D}|\Delta_\Gamma l^3)$*, where the* K*- shortest paths are computed and the largest one has length* l*, due to Lemma 7.3.2, Lemma 7.4.1 and Lemma 7.4.2.*

Algorithm 17 Single-path forwarding in source nodes

Input: A message MSG.
Input: Label of the source node $u \in L$.
Input: Label of the destination node $v \in L$.
1: Compute the shortest path $\widehat{w}$ from $\overline{u}$ to $\overline{v}$, where $w = x_1 x_2 \dots x_l$ (Algorithm 4).
2: **if** $l = 1$ **then**
3: Prepare a header $h \leftarrow \$$.
4: **else**
5: Prepare a header $h \leftarrow x_2 \dots x_l$.
6: Attach h to MSG.
7: Set the output $p \leftarrow x_1$.
8: Forward MSG through p.

Algorithm 18 Multi-path forwarding in source nodes

Input: A set of messages messages $\texttt{MSG}_1, \texttt{MSG}_2, \dots, \texttt{MSG}_m$.
Input: Label of the source node $u \in L$.
Input: Label of the destination node $v \in L$.
1: Compute a set of m paths $\widehat{w}_1, \widehat{w}_2, \dots \widehat{w}_m$ from $\overline{u}$ to $\overline{v}$, where $w_i = x_{i,1} x_{i,2} \dots x_{i,l_i}$ (Algorithm 7 / Algorithm 8 / Algorithm 9).
2: **for each** $\texttt{MSG}_i$ **do**
3: **if** $l_i = 1$ **then**
4: Prepare a header $h_i \leftarrow \$$.
5: **else**
6: Prepare a header $h_i \leftarrow x_{i,2} \dots x_{i,l_i}$.
7: Attach each h_i to $\texttt{MSG}_i$.
8: Set the output port $p_i \leftarrow x_{i,1}$.
9: Forward $\texttt{MSG}_i$ through p_i.

Algorithm 19 Forwarding in intermediate nodes

Input: A message MSG.
1: Read the header $h' = y_1 y_2 \dots y_j$ of MSG.
2: **if** $h' = \$$ **then**
3: Finish.
4: **if** $j > 1$ **then**
5: Prepare a new header $h \leftarrow y_2 \dots y_j$.
6: **else**
7: Prepare a new header $h \leftarrow \$$.
8: Replace the old header h' by the new one h in MSG.
9: Set the output $p \leftarrow y_1$.
10: Forward MSG through p.

Lemma 9.1.4 *The forwarding decision in mode deterministic of the WPR is taken by intermediate nodes using Algorithm 19, which runs in time* $O(1)$ *due to the next node in the path is encoded in the message header and thus no path is computed.*

9.2 FAULT-TOLERANT ROUTING

To provide fault-tolerance, the WPR applies hop-by-hop routing in mode single-path. The routing path is computed at the source node and if it is necessary also at some intermediate nodes. Every node keeps records of link and node failures, and uses Algorithm 10 to compute the routing path avoiding the recorded failures. Since nodes keep a partial record of failures, it is possible that the computed path contains failures that were not recorded. However, what it is important in hop-by-hop routing is that the next node in the computed path is correct.

Before explaining the forwarding protocol, let us present the types of message headers. A header consists of a string with at most $D_\Gamma + 1$ symbols. The first symbol in the header must be $\widehat{h}$ or $\overline{h}$. For headers beginning with $\widehat{h}$, the remaining substring denotes the path from the node receiving the message to the destination node. For headers beginning with $\overline{h}$, the remaining substring denotes the label of the destination node.

Lemma 9.2.1 *A message header defined by the WPR, in fault-tolerant mode, can be represented with* $O(D_\Gamma \log(\Delta_\Gamma))$ *bits.*

Proof. In fault-tolerant mode, the WPR defines two kinds of headers. The first one is given by a word representing a path of length less o equal to the network diameter. The second one is given by the label of the destination node. Both headers can be represented with $O(D_\Gamma \log(\Delta_\Gamma))$ bits. The proof is analogous to the proof of Lemma 6.1.1. ■

Let us now present Algorithm 20, which describe the process of fault-tolerant forwarding in source nodes. Consider a source node $\overline{u}$ that wants to send a message `MSG` to another node $\overline{v}$. First, the shortest path to $\overline{v}$ avoiding the recorded failures is computed, i.e. $w = x_1 x_2 \dots x_l$, (line 1). If there is no path to $\overline{v}$, the algorithm ends (*if* condition at line 2). Otherwise, a header is prepared according to the size of the computed path according to the following cases:

Case 1: The computed path has length 1 (line 4).
In this case, the header is $\overline{h} \cdot \$$. It indicates to the next node in the path that is the destination node.

Case 2: The computed path is larger than 1 but shorter than the network diameter (line 6).
In this case, the header is $\widehat{h} \cdot x_2 \dots x_l$. The next in the path may or may not compute the remaining routing path to the destination node depending on whether there are failures in the network or not.

Case 3: The computed path is larger than the network diameter (line 8).
In this case, the header is $\overline{h} \cdot v$. It avoid to have headers with more that $D_\Gamma + 1$ symbols However, the next node in the path must compute the remaining routing path to the destination node.

Finally, the header is attached to MSG and it is forwarded towards the next node in the path (lines 10-12).

Algorithm 20 Fault-tolerant forwarding in source nodes

Input: A message MSG.
Input: Label of the source node $u \in L$.
Input: Label of the destination node $v \in L$.
1: Compute the shortest path $\widehat{w}$ from $\overline{u}$ to $\overline{v}$ avoiding nodes in $\mathcal{V}_u$ and links $\mathcal{E}_u$, where $w = x_1 x_2 \dots x_l$ (Algorithm 10).
2: **if** $w = Null$ **then**
3: Finish.
4: **if** $l = 1$ **then**
5: Prepare a header $h \leftarrow \widehat{h} \cdot \$$.
6: **else if** $1 < l \leqslant D_\Gamma$ **then**
7: Prepare a header $h \leftarrow \widehat{h} \cdot x_2 \dots x_l$.
8: **else if** $l > D_\Gamma$ **then**
9: Prepare a header $h \leftarrow \overline{h} \cdot v$.
10: Attach h to MSG.
11: Set the output $p \leftarrow x_1$.
12: Forward MSG through p.

Lemma 9.2.2 *The forwarding decision in fault-tolerant mode of the WPR is taken by source nodes employing Algorithm 20, which execute Algorithm 10 to compute the routing path. Then Algorithm 20 runs in time* $O(K|\mathbb{D}|\Delta_\Gamma l^3)$*, where the shortest path avoiding failures is the* K*-th shortest path in the network without failures and has length* l*, due to Lemma 7.5.1.*

Turning now to present Algorithm 21, which describe the process of fault-tolerant forwarding in intermediate nodes, consider an intermediate node $\overline{u}$ that receives a message MSG with header h'. First, it is checked whether $\overline{u}$ is the destination node. If so, the routing process ends (*if* condition at line 2). Otherwise, a new header is prepared and the output port is identified according to the following two cases:

Case 1: h' consists of the routing path and there is no failures recorded (line 4).
In this case, the routing path and output port are given by h'.

Case 2: h' consists of the label of the destination node or there are failures recorded (line 10).
In this case, the label of the destination node is computed (*if* conditions at lines 11 and 13) and the algorithm proceeds similarly to Algorithm 20.

Finally, the old header h' is replaced by the new one h in MSG and it is forwarded towards the next node in the path (lines 25-26).

Lemma 9.2.3 *The forwarding decision in fault-tolerant mode of the WPR is taken by intermediate nodes employing Algorithm 21. If the routing path is given by the message header, then no path is computed and thus the algorithm runs in time* $O(1)$*. If the routing path needs to be computed, then it is executed Algorithm 10 to compute it. Therefore, the algorithm runs in time* $O(K|\mathbb{D}|\Delta_\Gamma l^3)$*, where the*

Algorithm 21 Fault-tolerant forwarding in intermediate nodes

Input: A message MSG.
Input: Label of the intermediate node $u \in L$.
Read the header $h' = y_1 y_2 \dots y_j$ from MSG.
if $h' = \widehat{h} \cdot \$$ **then**
  Finish.
if $y_1 = \widehat{h}$ AND $\mathcal{V}_u = \emptyset$ AND $\mathcal{E}_u = \emptyset$ **then**
  if $j = 2$ **then**
    Prepare a new header $h \leftarrow \widehat{h} \cdot \$$.
  else if $j > 2$ **then**
    Prepare a new header $h \leftarrow \widehat{h} \cdot y_3 \dots y_j$.
  Set the output $p \leftarrow y_2$.
else if $y_1 = \overline{h}$ OR $\mathcal{V}_u \neq \emptyset$ OR $\mathcal{E}_u \neq \emptyset$ **then**
  if $y_1 = \overline{h}$ **then**
    $v \leftarrow y_2 \dots y_j$.
  else if $y_1 = \widehat{h}$ **then**
    $v \leftarrow$ the canonical form of $u \cdot y_3 \dots y_j$ (Algorithm 1).
  Compute the shortest path $\widehat{w}$ from $\overline{u}$ to $\overline{v}$ avoiding nodes in $\mathcal{V}_u$ and links $\mathcal{E}_u$, where $w = x_1 x_2 \dots x_l$ (Algorithm 10).
  if $w = Null$ **then**
    Finish.
  if $l = 1$ **then**
    Prepare a new header $h \leftarrow \widehat{h} \cdot \$$.
  else if $1 < l \leqslant D_\Gamma$ **then**
    Prepare a new header $h \leftarrow \widehat{h} \cdot x_2 \dots x_l$.
  else if $l > D_\Gamma$ **then**
    Prepare a new header $h \leftarrow \overline{h} \cdot v$.
  Set the output $p \leftarrow x_1$.
Replace the old header h' by the new one h in MSG.
Forward MSG through p.

shortest path avoiding failures is the K*-th shortest path in the network without failures and has length* l*, due to Lemma 7.5.1.*

What need you flow so fast?

— Anonymous

10 COMPLEXITY ANALYSIS

This chapter presents a complexity analysis of the WPR. The analysis is based on the complexity measures presented in Section 1.2.3 and follows the notation presented in Table 14. This chapter proceeds as follow. First, a space complexity analysis of the WD automaton is presented. These results determine the space and time complexity of the WPR. After that, the analyses regarding space and time complexity of the WPR are presented. It involves an evaluation on the families of CGs introduced in Table 2. Then, complexities of the distributed processes employed by the WPR are summarized. Finally, it is presented a comparison between the WPR and the state of the art on routing in CGs that is presented in Chapter 5.

10.1 THE WORD-DIFFERENCE AUTOMATON

This section presents a space complexity analysis of the WD automaton, i.e. *Diff*. This automaton is of major importance in the operation of the WPR and their path computation algorithms as it encodes the structure of its related CG (see Section 6.1.3). as it is shown in the following subsections, the space and time complexity of the WPR depends on the space complexity of *Diff* and its number of states, i.e. $|\mathbb{D}|$.

Lemma 10.1.1 *The space complexity of the WD automaton, i.e. Diff, depends on the value of the fellow-traveler constant of its related CG, i.e.* k*, as follows: 1)* $O(\Delta_\Gamma^3 \log(\Delta_\Gamma))$ *if* $k = 1$*, 2)* $O(\Delta_\Gamma^{k+1} \log(\Delta_\Gamma))$ *if* $1 < k < D_\Gamma$*, and 3)* $O(n\Delta_\Gamma^2 \log(\Delta_\Gamma))$ *if* $k \approx O(D_\Gamma)$.

Proof. From Definition 4.4.3, the set of states of *Diff* is given by $\mathbb{D}$ and its set of transitions is given by tuples in B. In the worst case, the number of transitions of *Diff* is $|\mathbb{D}||B|$, where $|B| \approx O(\Delta_\Gamma^2)$. Since each transition can be represented by $O(\log(\Delta_\Gamma))$ bits, the set of transitions can be represented by $O(|\mathbb{D}|\Delta_\Gamma^2 \log(\Delta_\Gamma))$ bits. From Definition 4.4.2, $\mathbb{D}$ is given by the labels of nodes in the k-neighbourhood of node $\overline{\$}$, i.e. $N(\overline{\$}, k)$, see Definition 2.1.11. Thereby the space complexity of *Diff* depends on the value of k as follows:

1) If $k = 1$, then $\mathbb{D} = A \cup \{\$\}$ and $|\mathbb{D}| = \Delta_\Gamma + 1$. Thus $\mathbb{D}$ can be represented by $O(\Delta_\Gamma \log(\Delta_\Gamma))$ bits and the set of transitions can be represented by $O(\Delta_\Gamma^3 \log(\Delta_\Gamma))$ bits. Therefore, the space complexity of *Diff* is $O(\Delta_\Gamma^3 \log(\Delta_\Gamma))$.

2) If $1 < k < D_\Gamma$, then, in the worst case, $|\mathbb{D}| \approx O(\sum_{l=0}^{k} \Delta_\Gamma^l) = O(\Delta_\Gamma^{k-1})$. Thus $\mathbb{D}$ can be represented by $O(\sum_{l=0}^{k} l\Delta_\Gamma^l \log(\Delta_\Gamma)) = O(k\Delta_\Gamma^k \log(\Delta))$ bits and the set of transitions can be represented by $O(\Delta_\Gamma^{k+1} \log(\Delta_\Gamma))$ bits. Therefore, the space complexity of *Diff* is $O(\Delta_\Gamma^{k+1} \log(\Delta_\Gamma))$.

3) If $k \approx O(D_\Gamma)$, then $\mathbb{D} = L$ and $|\mathbb{D}| \approx O(n)$. Thus $\mathbb{D}$ can be represented by $O(D_\Gamma \log(\Delta_\Gamma) \sum_{l=1}^{D_\Gamma} l) = O(D_\Gamma^3 \log(\Delta_\Gamma))$ bits and the set of transitions can be represented by $O(n\Delta_\Gamma^2 \log(\Delta_\Gamma))$ bits. Therefore, the space complexity of *Diff* is $O(n\Delta_\Gamma^2 \log(\Delta_\Gamma))$.

■

From Lemma 10.1.1, the size of *Diff* depends on the value of k and the cadinality of $\mathbb{D}$. More concretely, it depends just on the value of k as $\mathbb{D}$ is defined in terms of k, i.e. $\mathbb{D}$ is given by the labels of nodes in $N(\overline{\$}, k)$. In [32], it is shown that the value of k is bounded by the hyperbolicity, i.e. δ of the CG, according to the relation $\delta > \frac{k}{2}$, where $\delta \leqslant \frac{D_\Gamma}{2}$, then $k \leqslant D_\Gamma$. as it is explained in Lemma 10.1.1, if $k = D_\Gamma$, then $\mathbb{D}$ represents the whole network graph. Nevertheless, the value of k was calculated through computer simulations for the families of CGs introduced in Table 2. Table 17 shows the upper-bounds of the value of k and the cardinality of $\mathbb{D}$ for evaluated families of CGs. as it is shown, the value of k is bounded in the 2D Torus, Hypercube, Transposition and Bubble-sort graphs, whereas in the Star and Butterfly graphs, k grows proportionally in terms of D_Γ. Actually the value of k for Butterfly graphs is very close to D_Γ.

It is important to mention that different lexicographic orders may result in *Diff* of different sizes (see [37] Section 13.2.1). It is unknown which lexicographic order leads to the smallest *Diff*. The results presented in Table 17 were obtained by using the lexicographic order given by the definitions of the generating sets of CGs introduced in Section 3.3.1.

Table 17: Estimate of the *fellow-traveler* constant and the cardinality of the set of word-differences.

Cayley graph family	k	$\lvert\mathbb{D}\rvert$
2D Torus	2	$O(1)$
Hypercube		$O(\log(n))$
Bubble-sort	4	$O(\log(n)^3)$
Star	$O(D_\Gamma)$	$O(n)$
Transposition	2	$O(\log(n))$
Butterfly	$O(D_\Gamma)$	$O(n)$

10.2 SPACE COMPLEXITY

This section presents results about space complexity of the WPR. In order to compare the WPR and its path computation algorithms with the state of the art on routing for CG, the results are organized as follows: memory space requirements per node to maintain the WPR, space complexity of the path computation algorithms (which runs in each node) and space complexity of message headers.

From Section 6.1, the routing information required by each node to maintain the WPR consists of: 1) a unique node label; 2) a routing table; and 3) the WD automaton. Among these information, the path computation algorithms require only the WD automaton. Regarding to message headers, they consist of: a word representing a *ShortLex* path if the routing process is deterministic, or a word representing either a shortest path or a node label if the routing process is fault-tolerant. Table 18 summarizes the memory space requirements of the WPR. as it is shown, node labels and message headers have the same size, which is determined by Δ_Γ and D_Γ. The size of routing tables depends only on Δ_Γ. Finally, the size of the WD automaton depends on Δ_Γ and k.

Table 19 summarizes the space complexity of the WPR working on the families of CGs introduced in Table 2. Note that the information presented in Table 19 is organized according to the routing information introduced in Table 18. Regarding the memory space requirements per node, Hypercube graphs have the lowest ones as they have the lowest values of Δ_Γ, D_Γ and k. Conversely, the Star and Butterfly graphs have the highest memory space requirements per node and the highest space complexity for the path computation algorithms algorithms. The reason is that these families of CGs have a value of k that grows proportionally to D_Γ and then the size of the WD automaton is close to the size of the whole CG. Turning now to 2D Torus graphs, these CGs also have high memory space requirements per node, i.e. $O(n^{1/2})$, which is determined by their value of D_Γ. However, the space complexity of the path computation algorithms on 2D Torus graphs is constant as its values of k and Δ_Γ are constant. Finally, in Bubble-sort and Transposition graphs, the memory space requirements per node and space complexity of the path computation algorithms are polylogarithmic due to their values of Δ_Γ and D_Γ are also polylogarithmic and their values of k are constant. Regarding to the space complexity of the message headers (and node labels), it is polylogarithmic in all the analyzed families of CGs except 2D Torus graphs, which have the largest message headers.

10.3 TIME COMPLEXITY

This section presents a summary of the forwarding decision time complexity of the WPR, and time complexity of their path computation algorithms. These algorithms and their time complexity analysis are presented in Chapter 7. The forwarding protocols and their time complexity analysis are presented in Chapter 9. Table 20 summarizes the time complexity of the path computation algorithms. The value of K indicates the number of paths that are computed, meanwhile l indicates the length of the largest

Table 18: Space complexity of the Word-Processing-based Routing.

Entity	Routing information	Space complexity	Proof
Node	Node label	$O(D_\Gamma \log(\Delta_\Gamma))$	Lemma 6.1.1
	Routing table	$O(\Delta_\Gamma \log(\Delta_\Gamma))$	Preposition 6.1.1
Path computation algorithms	Word-difference automaton	$O(\Delta_\Gamma^3 \log(\Delta_\Gamma))$, if $k = 1$ $O(\Delta_\Gamma^{k+1} \log(\Delta_\Gamma))$, if $1 < k < D_\Gamma$ $O(n\Delta_\Gamma^2 \log(\Delta_\Gamma))$, if $k \approx O(D_\Gamma)$	Lemma 10.1.1
Message	Message header	$O(D_\Gamma \log(\Delta_\Gamma))$	Lemma 9.1.1, Lemma 9.2.1

Table 19: Space complexity of the Word-Processing-based Routing on specific families of Cayley graphs.

Cayley graph family	Node label / message header	Routing table	Word-difference automaton (path computation algorithms)
2D Torus	$O(n^{1/2})$	$O(1)$	$O(1)$
Hypercube	$O(\log(n)\log(\log(n)))$	$O(\log(n)\log(\log(n)))$	$O(\log(n)^3\log(\log(n)))$
Bubble-sort	$O(\log(n)^2\log(\log(n)))$		$O(\log(n)^5\log(\log(n)))$
Star	$O(\log(n)\log(\log(n)))$		$O(n\log(n)^2\log(\log(n)))$
Transposition		$O(\log(n)^2\log(\log(n)))$	$O(\log(n)^6\log(\log(n)))$
Butterfly	$O(\log(n))$	$O(1)$	$O(n)$

path that is computed (sometimes $l = D_\Gamma$). Recall from Chapter 7 that paths are sequentially computed from the shortest path until all the required paths are computed. Therefore, the shortest path is computed at the beginning of all the path computation algorithms, and then the time complexity of the shortest path algorithm (Algorithm 4) is included into the time complexity of the remaining algorithms. In addition to K, l and D_Γ, the time complexity depends on $|\mathbb{D}|$ and Δ_Γ. The reason is that paths are searched by exploring the WD automaton, whose states and transitions are given by $\mathbb{D}$ and Δ_Γ, respectively.

Table 20: Time complexity of the path computation algorithms of the Word-Processing-based Routing.

Algorithm	Computed paths	Time complexity	Proof
Algorithm 4	The shortest path	$O(\lvert\mathbb{D}\rvert\Delta_\Gamma D_\Gamma^2)$	Lemma 7.2.1
Algorithm 5	The minimal paths	$O(\max\{K, D_\Gamma\}\lvert\mathbb{D}\rvert\Delta_\Gamma D_\Gamma)$	Lemma 7.2.3
Algorithm 7	The K-shortest paths	$O(\max\{Kl, D_\Gamma^2\}\lvert\mathbb{D}\rvert\Delta_\Gamma)$	Lemma 7.3.2
Algorithm 8	The link-disjoint paths	$O(K\lvert\mathbb{D}\rvert\Delta_\Gamma l^3)$	Lemma 7.4.1
Algorithm 9	The node-disjoint paths		Lemma 7.4.2
Algorithm 10	The shortest path avoiding a set of nodes and links		Lemma 7.5.1

Table 21 presents the forwarding decision time complexity of the WPR. Recall from Section 1.2.2.3 that in source routing, routing paths are computed just in source nodes, hence the forwarding decision time is constant in intermediate nodes. Conversely, in hop-by-hop routing, routing paths are computed along all nodes in the path. If the forwarding decision involves the computation of routing paths, then some of the algorithms presented in Table 20 are employed. The selection of the algorithm depends on the adaptability and path diversity required.

Table 22 presents the time complexity of the path computation algorithms running in the families of CGs presented in Table 2. Then, this table also provides the complexity of the forwarding decision time as it is defined by the path computation algorithms. The 2D Torus, Star and Butterfly graphs have the highest time complexity in all the algorithms. In the case of 2D Torus graphs, the high time complexity is a consequence of their high value of D_Γ. In the case of Star and Butterfly graphs, their high value of k results in a large $|\mathbb{D}|$ and thus in high time complexity. Regarding Hypercube, Bubble-sort and Transposition graphs, the time complexity of all

Table 21: Forwarding decision time complexity of the Word-Processing-based Routing.

Routing decision	Node	Adaptability	Path diversity	Path computation algorithm	Time complexity	Proof
Source routing	Source	Deterministic	Single-path	Algorithm 4	$O(\lvert\mathbb{D}\rvert\Delta_r D_r^2)$	Lemma 9.1.2
			Link-disjoint paths	Algorithm 8	$O(\kappa\lvert\mathbb{D}\rvert\Delta_r l^3)$	Lemma 9.1.3
			Node-disjoint paths	Algorithm 9		
	Intermediate		Single-path	N/A	$O(1)$	Lemma 9.1.4
			Link-disjoint paths			
			Node-disjoint paths			
Hop-by-hop routing	Source and intermediate	Fault-tolerant	Single-path	Algorithm 10	$O(\kappa\lvert\mathbb{D}\rvert\Delta_r l^3)$	Lemma 9.2.2, Lemma 9.2.3

the algorithms is polylogarithmic due to Δ_Γ and D_Γ have polylogarithmic growth, and the value of k is constant and then $|\mathbb{D}|$ also have polylogarithmic growth. All the path computation algorithms have their lowest time complexity in Hypercube graphs due to their low values of Δ_Γ, D_Γ and k.

10.4 COMPLEXITY OF DISTRIBUTED PROCESSES

The WPR uses the following distributed processes: 1) node label assignment (Section 6.3), 2) (node/link) failure notification (Section 8.3), and 3) (node/link) recovery notification (Section 8.4). Table 23 summarizes the convergence time and message complexity of these processes. In the process of node label assignment, the CG is explored through Breadth-First Search (BFS). When a node is visited, it computes its label, which involves the computation of a shortest path (Algorithm 4). Then, the convergence time is proportional to D_Γ and to the time complexity of the Algorithm 4. After a node computes its label, it forwards $\Delta_\Gamma - 1$ messages (except the root node, which sends Δ_Γ messages) to continue the exploration.

Regarding the processes of failure and recovery notification, the notified nodes are those in the neighbourhood of the faulty or recovery node $\bar{f}$ given by $N(\bar{f}, \lceil \frac{c}{2} \rceil)$ (see Lemma 8.3.1, Lemma 8.3.2, Lemma 8.4.1 and Lemma 8.4.2). Then message complexity is proportional to the cardinality of this neighbourhood. Each notified node decides if update its records of failures, which involves to execute the algorithm for computing paths avoiding nodes and links (Algorithm 10). In the worst case, each notified node executes Algorithm 10 once by each failure in the network, i.e. $|V_u| + |E_u|$.

10.5 COMPARISON WITH THE STATE OF THE ART PROPOSALS

This section presents a comparison between the WPR and the state of the art on routing in CGs presented in Chapter 5. First, the path computation algorithms of the WPR (summarized in Table 20) are compared with the algorithms introduced in Section 5.1. Then the WPR is compared with the routing schemes presented in Section 5.2.

Table 24 compares the paths computed by the analyzed algorithms. as it is shown, all the algorithms compute the shortest path. In addition the SFA computes the minimal paths, and the PCAACG computes the disjoint paths. Regarding the algorithms of the WPR, they not only compute the aforementioned paths but also the K-shortest paths. It is important to mention that a thorough search of the relevant literature did not yield any generic algorithm for computing the K-shortest paths, which is fundamental to design routing schemes that provide path diversity and fault-tolerance.

Table 25 indicates which of the analyzed algorithms have the best space and time complexity when working on the families of CG presented in Table 2. These results are obtained from a comparison of the space and time complexity of the SFA, the PCAACG (see Table 6 and Table 7) and the algorithms of WPR (see Table 19 and Table 20). The PCAACG is considered just in 2D Torus and Hypercube graphs as these are the only abelian CGs among the evaluated CGs.

Table 22: Time complexity of the path computation algorithms of the Word-Processing-based Routing on specific families of Cayley graphs.

	Computed paths			
Cayley graph family	**The shortest path**	**The minimal paths**	**The K-shortest paths**	**Disjoint paths / The shortest path avoiding nodes and links**
2D Torus	$O(n)$	$O(\max\{K, n^{1/2}\}n^{1/2})$	$O(\max\{Kl, n\})$	$O(Kl^3)$
Hypercube	$O(\log(n)^4)$	$O(\max\{K, \log(n)\}\log(n)^3)$	$O(\max\{Kl, \log(n)^2\}\log(n)^2)$	$O(K\log(n)^2l^3)$
Bubble-sort	$O(\log(n)^8)$	$O(\max\{K, \log(n)^2\}\log(n)^6)$	$O(\max\{Kl, \log(n)^4\}\log(n)^4)$	$O(K\log(n)^4l^3)$
Star	$O(n\log(n)^3)$	$O(\max\{K, \log(n)\}n\log(n)^2)$	$O(\max\{Kl, \log(n)^2\}n\log(n))$	$O(Kn\log(n)l^3)$
Transposition	$O(\log(n)^5)$	$O(\max\{K, \log(n)\}\log(n)^4)$	$O(\max\{Kl, \log(n)^2\}\log(n)^3)$	$O(K\log(n)^3l^3)$
Butterfly	$O(n\log(n)^2)$	$O(\max\{K, \log(n)\}n\log(n))$	$O(\max\{Kl, \log(n)^2\}n)$	$O(Knl^3)$

Table 23: Complexity measures of the distributed processes used by the Word-Processing-based Routing.

Distributed process	Convergence time	Message complexity	Proof
Node label assignment	$O(\lvert\mathbb{D}\rvert\Delta_\Gamma D_\Gamma^3)$	$n(\Delta_\Gamma - 1) + 1$	Lemma 6.3.1
Failure / recovery notification	$O(\max\{1, \lvert V_\Gamma\rvert, \lvert E_\Gamma\rvert\}$ $K\lvert\mathbb{D}\rvert\Delta_\Gamma^2 \lceil \frac{c}{2} \rceil^4)$	$O(\Delta_\Gamma \lvert N(\bar{f}, \lceil \frac{c}{2} \rceil)\rvert)$	Lemma 8.3.1, Lemma 8.3.2, Lemma 8.4.1, Lemma 8.4.2

Table 24: Paths computed by generic algorithms on Cayley graphs.

Algorithm	Group family	The shortest path	The minimal paths	The K-shortest paths	The disjoint paths
Algorithms of the WPR	Automatic groups	Yes	Yes	Yes	Yes
SFA	Any group	Yes	Yes	No	No
PCAACG	Abelian groups	Yes	No	No	Yes

From Table 25, the PCAACG has the best space and time complexity in the cases, where it can be applied. i.e. in the computation of the shortest path and the disjoint paths on 2D Torus and Hypercube graphs. Regarding SFA, it can work on all the evaluated families of CGs to compute the shortest path and the minimal paths. In these cases, the SFA has the best space complexity in three of the six evaluated families of CGs, and reaches the best time complexity in four of them. Regarding the algorithms of the WPR, they outperform the space complexity of SFA in Bubble-sort and Transposition graphs; outperform the time complexity in the computation of the shortest path in Transposition graphs; and outperform the time complexity in the computation of the minimal paths in Hypercubes and Transposition graphs. Finally, among the evaluated algorithms, the algorithms of the WPR are the only ones able to compute the K-shortest paths (Algorithm 5) and the disjoint paths (Algorithm 9, Algorithm 8) in non-abelian CGs.

Table 25: Generic path computation algorithms with best space and time complexity on specific families of Cayley graphs.

Cayley graph family	Best space complexity	Best time complexity: The shortest path	Best time complexity: The minimal paths	Best time complexity: The K-shortest paths	Best time complexity: The disjoint paths
2D Torus	SFA / PCAACG	SFA / PCAACG	SFA	Algorithms of the WPR	PCAACG
Hypercube	PCAACG	PCAACG	Algorithms of the WPR		Algorithms of the WPR
Bubble-sort	Algorithms of the WPR	SFA	SFA		
Star	SFA				
Transposition	Algorithms WPR	Algorithms WPR	Algorithms WPR		
Butterfly	SFA	SFA	SFA		

Based on the objectives and features of routing, which are presented in Section 1.2.2.2 and Section 1.2.2.1, Table 26 presents a comparison of the analized routing schemes.The RPS and the GRWMS are deterministic and do not provide path diversity. However, these routing schemes provide minimal routing and guarantee packet delivery. The WPR and the RCRR are fault-tolerant. In addition, the WPR provides minimal routing and guarantees packet delivery, while the RCRR, in contrast, does not. Hence, the WPR is the most robust routing scheme among the analyzed schemes.

Table 26: Features of generic routing schemes for Cayley graphs.

Routing scheme	Minimal routing	Path diversity	Fault-tolerant	Packet delivery is guaranteed
WPR	Yes	Yes	Yes	Yes
RPS		No	No	
RCRR	No	No	Yes	No
GRWMS	Yes		No	Yes

Table 27 indicates which of the analyzed routing algorithms have the best space complexity when working on the families of CGs introduced in Table 2. These results are obtained from the comparison of the space complexity of RPS, the RCRR, the GRWMS (see Table 10, Table 11) and the WPR (see Table 19). Regarding message header, the WPR and GRWMS have the best space complexity in all the evaluated families of CGs except for Bubble-sort graphs. On the other hand, the RPS has the best space complexity in four of the six evaluated families of CGs. Turning now to the routing information per nodes, the RPS has the best space complexity per node in all the evaluated families of CGs except for 2D Torus. Finally, the WPR has the best space complexity per node in 2D Torus and Star graphs. Note that the RCRR is not in Table 27 as it has the highest space complexity.

Regarding the forwarding decision time, the RCRR has the best time complexity in both deterministic and fault-tolerant routing. In contrast, it has the worst space complexity as it employs full routing tables. As result, its time complexity is constant in deterministic routing and polylogarithmic in fault-tolerant routing (see Table 13). The WPR also has polylogarithmic time in fault-tolerant routing when work on four of the six evaluated families of CGs (see Table 22) and, in contrast to the RCRR, the WPR guarantees the packet delivery. In deterministic routing, the WPR outperforms the forwarding decision time of the GRWMS.

Table 27: Generic routing schemes with best space complexity on specific families of Cayley graphs.

Cayley graph family	Message header	Routing information per node
2D Torus	WPR / GRWMS	WPR
Hypercube	WPR / RPS / GRWMS	RPS
Bubble-sort	RPS	
Star	WPR / RPS / GRWMS	WPR / RPS
Transposition		RPS
Butterfly	WPR / GRWMS	

11 CONCLUSIONS

11.1 SUMMARY OF COMPLETED WORK

This Thesis has addressed the problem of routing in CGs, which can be classified into generic routing, i.e. that works on several families of CGs, and routing specialized on a family of CGs. The problem was focused on generic routing, which involves a major challenge as generic routing schemes must work on CGs with different topological structure. In addition to routing schemes, path computation algorithms are considered as routing proposals due to they are a fundamental part of the routing process. After providing an overview of the general problem and the proposed objectives to tackle it, Part I presented an analysis of CGs which is divided into four chapters. Chapter 2 provided the necessary mathematical tools for analyzing CGs from an approach of Graph Theory and Group Theory. This latter part puts special emphasis on permutation groups as they provide an homogeneous representation of CGs to study them.

In Chapter 3, the topological properties of CGs were analyzed in order to understand how these properties enable high performance and robustness on communication networks that use CGs as topology. Specifically, it was shown that: node-transitivity allows the use of the same communication algorithms in each node; link-connectivity allows load balancing, tolerance for random link failures; and low average distance enable low latency and support for a high number of end points. Then, it was presented a analysis of performance and robustness of six well-known families of CGs that has been widely used as network topologies. This analysis was employed in Chapter 5 and Chapter 10 to compare the impact of CGs with different topological structures on the performance of routing schemes and path computation algorithms.

Chapter 4 presented a theoretical framework to solve the shortest path problem in CGs from the point of view of the AGT. This approach consists in representing CGs as regular languages that are defined by FSAs, and then word processing techniques are applied on such FSAs to solving the shortest path problem. The AGT approach was applied in the GRWMS to propose a deterministic routing scheme that computes the shortest paths. This dissertation has extended and enhanced the GRWMS At the end of Part I, Chapter 5 presented the state of the art for generic routing in CGs, which includes routing schemes and path computation algorithms. This review provides a complexity analysis of the routing proposals together with a comparison of them working on six well-known families of CGs.

Through five chapters, Part II introduced the WPR, which is the main contribution of this research work. Chapter 6 gave an overview of the WPR. First, it was presented the necessary routing information per node for the operation of the WPR, which consists of 1) a unique label given by a word, 2) a routing table given by an alphabet that identifies the output ports, and 3) an automaton that encodes topological structure of CG and is

manipulated by the path computation algorithms. Then, it was explained relation between the features of the WPR and its path computation algorithms. These features of the WPR are: minimal routing, source routing, hop-by-hop routing, fault-tolerance and path diversity. Finally, it was presented a distributed process for node label assignment together with its complexity analysis. It is the only configuration process required by the WPR.

Chapter 7 presented the set of path computation algorithms of the WPR together with their complexity analysis. These algorithms can be named according to the paths that compute, as follows: 1) the shortest path, 2) the minimal paths, 3) the K-shortest paths, 4) the link-disjoint paths, 5) the node-disjoint paths, and 6) the shortest path avoiding a set of nodes and links. The shortest path algorithm results from solving the MWP, which arises from AGT and is equivalent to the shortest problem in CGs. This approach is also applied by the GRWMS, where the MWP is referred but it is no provided an algorithm to solve it. The algorithms to compute the minimal paths, the K-shortest paths, and the link/node disjoint paths allows path diversity. Fault-tolerance is enabled by the algorithm to compute the shortest path avoiding a set of nodes and links. All the algorithms compute the shortest paths and give as results word(s) representing the computed path(s). It enables minimal and source routing.

Chapter 8 introduced the mechanism of fault-tolerance, which supports multiple failures of both nodes and links. In addition, it provides minimal routing paths in spite of nodes do not keep a global record of failures. According to this mechanism, every node in the network keeps partial records of faulty nodes and links. Then, nodes compute routing paths avoiding the recorded failures. Since a routing path could contain failures that were not recorded, the fault-tolerant mechanism applies hop-by-hop routing in mode single-path. This chapter presents the detailed definition of the following process: failure record, notification of faulty nodes/links, and notification of recovery node/links. This chapter presented the detailed definition of these processes together with their complexity analysis.

Chapter 9 presented the forwarding protocols of the WPR, which are divided into deterministic and fault-tolerant. Regarding deterministic routing, there were introduced forwarding protocols for single-path and multi-path in modes source routing and hop-by-hop routing. Regarding fault-tolerant routing, there was presented a forwarding protocol for single-path in mode hop-by-hop routing. All the forwarding protocols employ message header whose size is the same as node labels. The forwarding protocols were presented together with their complexity analysis.

In order to completely validate the WPR, Chapter 10 provided a complexity analysis. It included an evaluation on six well-known families of CGs and a comparison with the state of the art on routing proposals for CGs, which is presented in Chapter 5. The results show that the algorithms for computing disjoint paths outperform the state of the art on generic algorithms. For some families of CG, the algorithms for computing the shortest path and minimal paths also enhance the state of the art. Regarding to the WPR it stays competitive with respect to the state of the art.

11.2 REVIEW OF CONTRIBUTIONS

This Thesis has proposed a generic routing scheme for CGs, which led to develop new strategies of path computation and fault-tolerance in CGs that contribute to the current state of the art on routing in CGs. Such contributions have been presented and peer-reviewed along different Publications, and they can be summarized as follows:

An analysis of the topological properties of CG. This Thesis provides a study of the impact of topological properties of CGs on the performance and robustness of networks that used them as topology [CITS'18]. It contributed to the performance evaluation of the proposed routing scheme on specific families of CGs.

A theoretical framework to study and solve problems related to path computation and routing in CGs from an approach of AGT. A fundamental idea of the AGT states that CGs can be represented by regular languages, which are encoded into FSA. The developed framework allows to study and solve problems related to routing in CGs by representing CGs as languages and employing techniques of word processing. A first version of this framework was applied to develop a deterministic single path routing scheme for DCNs whose topologies could be defined by CGs [TNET'18]. This framework was extended to face the multi-path problem and was employed to develop new generic path computation algorithms for CGs [DAM'18].

A set of generic algorithms for path computation in CGs. Employing the theoretical framework above described, this Thesis has proved that all the paths in a CG of a SAG can be sequentially computed from the shortest to the largest one. Following this approach, a set of generic path computation algorithms were developed [DAM'18]. Specifically, the proposed algorithms compute: 1) the shortest path, 2) the minimal paths, 3) the paths of bounded length, 4) the K-shortest paths, 5) the disjoint paths, and 6) the shortest path avoiding a set of nodes and edges. Through a complexity analysis, it was proved that the algorithms for computing disjoint paths outperform the state of the art on generic algorithms. For some families of CG, the algorithms for computing the shortest path and minimal paths also enhance the state of the art on generic algorithms. Finally, a thorough search of the relevant literature suggests it was proposed the first generic algorithm for computing K-shortest in CG.

The Word-Processing-based Routing (WPR). It is a generic routing scheme for CGs that guarantees packet delivery and provides: minimal routing, path diversity and fault-tolerance. As far as this author known, the WPR if the first generic with these features. The core of the WPR are the path computation algorithms described above and a novel mechanism of fault-tolerance, which support multiple failures and provides minimal routing in spite of nodes do not keep a global record of failures. Through a space and time complexity analysis, it was shown that the WPR stays competitive with respect to the state of the art on generic routing in CGs.

11.3 FUTURE WORK

There are several future research lines on which this Thesis can be taken as a base. The future work can be gathered in three main aspects

Designing new network architectures based on CGs and the WPR. It involves:

- Proposing a formal definition of the WPR as a routing protocol.
- Analyzing CGs that are link-transitive and satisfy the Moore bound in order to design high performance topologies.
- Evaluate the throughput of CGs under different traffic scenarios.
- Evaluate the robustness of CGs under different failure scenarios.
- Proposing methods for incremental expansion of nodes in CGs.

Deploy the WPR in WSN. It involves to propose network embeddings of random regular graphs in CGs.

Applying the approach of AGT to design distributed algorithms for CGs that solve problems different from the routing problem, e.g. clustering, leader election etc.

APPENDIX

A

PORT LABEL ASSIGNMENT

From the definition of CG (Definition 2.2.12), edges in every $\Gamma(\mathcal{G}, S)$ represent generators in S. In the process of node label assignment of the WPR (Section 6.3), ports of each node are required to be label with a letter representing their corresponding generator. This section describes the process of port label assignment in a distributed way.

This process consists in discovering the cycles in $\Gamma(\mathcal{G}, S)$ corresponding to each relator in R, and then deriving the corresponding generator of each link. Recall from Section 2.2.3 that the relators R of a group $\mathcal{G} = \langle S, R \rangle$ represent sequences of generators that are equivalent to the identity element. Therefore, relators can be described by words representing cycles in $\Gamma(\mathcal{G}, S)$. Algorithm 22 presents the steps to carry out the port label assignment in a distributed way. This process consists of the following phases: 1) discovering cycles, 2) reporting cycles, and 3) labelling links.

The phase of discovering cycles begins at an arbitrary node r, which sends a message `DSVR_RELATORS` through each of its ports i (see Algorithm 23). These messages are propagated at distance d from r, where d is half the length of the largest relator. At the end of this phase, messages `DSRV_RELATORS` will have traversed all the paths beginning at r and representing each relator. Each of these messages keeps its path history, which is saved at nodes receiving a message `DSRV_RELATORS` for the first time.

The phase of reporting cycles begins at nodes v receiving a message `DSRV_RELATORS` by second time. A discovered cycle consists of the path histories of the first and second messages `DSRV_RELATORS` that arrive to v. Each node v sends a message `RPT_CYCLE` to r, which contains the sequence of port numbers associated to the discovered cycle.

The phase of labelling nodes begins at node r, when it receives a number of messages `RPT_CYCLE` equal to the number of relators (Algorithm 22). Then, node r must identify each reported cycle to each relator and label its ports according to this relation. For each relator, node r sends a message `LABEL_PORTS` through the port corresponding to such relator. Messages `LABEL_PORTS` contain the representations of a relator as a word and as a sequence of ports. Each message is propagated along the cycle associated to its relator. Then nodes receiving a message `LABEL_PORTS` label their ports according to the relator contained in the message.

Algorithm 22 Distributed assignment of port labels.

1: Select an arbitrary node r of $V(\Gamma)$.
2: Node r sets $letter \leftarrow \epsilon$ and $path \leftarrow \epsilon$. All other nodes set $letter \leftarrow$ *Null* and $path \leftarrow$ Null.
3: Node r initializes the process to discover relators (Algorithm 23).
4: Every other node reacts to incoming messages as follows:

Upon receiving a message DSVR_RELATORS($depth$, $path\_history$) through its port j **do**:

1: **if** $path\_to\_r = Null$ **then**
2: $path\_to\_r \leftarrow reverse(path\_history)$.
3: **else**
4: $relator \leftarrow concatente(path\_history, path)$.
5: $index \leftarrow |path\_history| - 1$.
6: Send out the message RPT_CYCLE($relator, index$) through port j.
7: **if** $|path\_history| < depth$ **then**
8: **for** each port i except for j **do**
9: $path\_history.append(i)$.
10: Send out the message DSVR_RELATORS($depth, path\_history$) through port i.

Upon receiving a message RPT_CYCLE($relator, index$) through its port j **do**:

1: **if** $index == 0$ **then**
2: Initialize the process of labelling ports (Algorithm 24).
3: **else**
4: $i \leftarrow relator[index]$.
5: Send out the message RPT_CYCLE($generators, index - 1$) through port i.

Upon receiving a message LABEL_PORTS($relator\_port, relator\_word, index$) through its port j **do**:

1: Label port j as $relator\_word[index]$.
2: **if** $index < |relator\_port| - 1$ **then**
3: $index \leftarrow index + 1$.
4: Label port $relatoy\_port[index]$ as $relator\_word[index]$.
5: Send out the message LABEL_PORTS($relator\_port, relator\_word, index, \mathcal{A}$) through port $relator\_port[index]$.
6: **if** there is one port without label **then**
7: Label the remaining port i with the remaining letter $a \in \mathcal{A}$.
8: Send out the message LABEL_PORTS($[i], [a], 0$) through port i.

Algorithm 23 Discovery relators.

Input: A set of words $relators\_words$ representing the relators of $\Gamma(\mathcal{G}, S)$.

1: $d \leftarrow$ length of the largest relator in $relators\_words$.
2: $depth \leftarrow \lceil \frac{d}{2} \rceil$.
3: **for** each port i **do**
4: $path\_history \leftarrow [i]$.
5: Send out the message DSVR_RELATORS($depth, i, path\_history$) through port i.

Algorithm 24 Label ports.

Input: An tuple of letters $(gen1, gen2)$.
Input: A bijective map ϕ that assigns each port to a letter in $\mathcal{A}$.
Input: A set of words over $\mathcal{A}$ that represents the relators of $\Gamma(\mathcal{G}, S)$ and is denoted by relators_words.

1: relators_ports $\leftarrow$ relators_ports $\cup \{$relator$\}$.
2: **if** |relators_words| = |relators_ports| **then**
3: Correlate relators_words to relators_ports.
4: **for** each correlated pair (relator_port, relator_word) **do**
5: Label port relator_port[0] as relator_word[0].
6: Send out the message `LABEL_PORTS`(relator_port, relator_word, 1) through port relator_port[0].

BIBLIOGRAPHY

[1] S. B. Akers and B. Krishnamurthy. "A group-theoretic model for symmetric interconnection networks." In: *IEEE Transactions on Computers* 38.4 (Apr. 1989), pp. 555–566. ISSN: 0018-9340. DOI: 10.1109/12.21148.

[2] M. C. Heydemann. *Cayley graphs and interconnection networks*. Ed. by Geňa Hahn and Gert Sabidussi. Dordrecht: Springer Netherlands, 1997, pp. 167–224. ISBN: 978-94-015-8937-6. DOI: 10.1007/978-94-015-8937-6.

[3] Chuanxiong Guo et al. "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers." In: *ACM SIGCOMM Conference on Data Communication*. Barcelona, Spain: ACM, 2009, pp. 63–74. ISBN: 978-1-60558-594-9.

[4] J. Kim, J. Balfour, and W. J. Dally. "Flattened Butterfly Topology for On-Chip Networks." In: *IEEE Computer Architecture Letters* 6.2 (2007), pp. 37–40. ISSN: 1556-6056. DOI: 10.1109/L-CA.2007.10.

[5] Jaewook Yu, Eric Noel, and K. Wendy Tang. "A graph theoretic approach to ultrafast information distribution: Borel Cayley graph resizing algorithm." In: *Computer Communications* 33.17 (2010), pp. 2093–2104. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2010.07.013.

[6] J. X. Zhou, Z. L. Wu, S. C. Yang, and K. W. Yuan. "Symmetric Property and Reliability of Balanced Hypercube." In: *IEEE Transactions on Computers* 64.3 (Mar. 2015), pp. 876–881. ISSN: 0018-9340. DOI: 10.1109/TC.2014.2304391.

[7] Anthony H. Dekker and Bernard D. Colbert. "Network Robustness and Graph Topology." In: *27th Australasian Conference on Computer Science*. Vol. 26. ACSC. Dunedin, New Zealand: Australian Computer Society, Inc., 2004, pp. 359–368.

[8] Iain A. Stewart and Alejandro Erickson. "The influence of datacenter usage on symmetry in datacenter network design." In: *The Journal of Supercomputing* (2017). ISSN: 1573-0484. DOI: 10.1007/s11227-017-2217-1.

[9] Derek Robinson. *A Course in the Theory of Groups*. Graduate Texts in Mathematics 80. Second edition, 1996. Springer-Verlag, 1982. ISBN: 0-38790-600-2.

[10] B. Bollobas. *Random Graphs*. Cambridge University Press, 2001.

[11] L. G. Valiant. "A Scheme for Fast Parallel Communication." In: *SIAM Journal on Computing* 11.2 (1982), pp. 350–361. DOI: 10.1137/0211027.

[12] A. Singh. "Load-balanced Routing in Interconnection Networks." PhD thesis. Stanford University - Department of Electrical Engineering, 2005.

[13] David B. A. Eppstein. "Finding the k Shortest Paths." In: *SIAM Journal on Computing* 28.2 (1998), pp. 652–673. DOI: 10.1137/S0097539795290477.

[14] Jin Y. Yen. "Finding the K Shortest Loopless Paths in a Network." In: *Management Science* 17.11 (1971), pp. 712–716. DOI: 10.1287/mnsc.17.11.712.

[15] M. Camelo, D. Papadimitriou, L. Fàbrega, and P. Vilà. "Geometric Routing With Word-Metric Spaces." In: *Communications Letters, IEEE* 18.12 (2014), pp. 2125–2128. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2014.2364213.

[16] K. Wendy Tang and Bruce W. Arden. "Vertex-transitivity and Routing for Cayley Graphs in GCR Representations." In: *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing: Technological Challenges*. SAC '92. Kansas City, Missouri, USA: ACM, 1992, pp. 1180–1187. ISBN: 0-89791-502-X.

[17] Stephen T. Schibell and Richard M. Stafford. "Processor interconnection networks from Cayley graphs." In: *Discrete Applied Mathematics* 40.3 (1992), pp. 333 –357. ISSN: 0166-218X.

[18] Cheng Lai. "On the construction of all shortest vertex-disjoint paths in Cayley graphs of abelian groups." In: *Theoretical Computer Science* 571 (2015), pp. 10 –20. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2014.12.023.

[19] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000. ISBN: 0-89871-464-8.

[20] Dennis Abts and John Kim. *High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities*. Morgan & Claypool Publishers, 2011. ISBN: 978-1-608-45402-0.

[21] Dieter Jungnickel. *Graphs, Networks and Algorithms*. 3rd. Springer Publishing Company, Incorporated, 2007. ISBN: 3-540-72779-5.

[22] G. D. Stamoulis and J. N. Tsitsiklis. "The efficiency of greedy routing in hypercubes and butterflies." In: *IEEE Transactions on Communications* 42.11 (1994), pp. 3051–3061. ISSN: 0090-6778. DOI: 10.1109/26.328987.

[23] K. Tang and B. Arden. "Representations of Borel Cayley Graphs." In: *SIAM Journal on Discrete Mathematics* 6.4 (1993), pp. 655–676. DOI: 10.1137/0406050.

[24] Derek F. Holt, Sarah Rees, and Claas E. Röver. *Groups, Languages and Automata*. London Mathematical Society Student Texts. Cambridge University Press, 2017. DOI: 10.1017/9781316588246.

[25] David B. A. Epstein et al. *Word Processing in Groups*. Natick, MA, USA: A. K. Peters, Ltd., 1992. ISBN: 0-86720-244-0.

[26] Richard J Trudeau. *Introduction to graph theory*. Dover Books on Mathematics. Mineola, NY: Dover, 1994.

[27] Ashwin Ganesan. "Cayley graphs and symmetric interconnection networks." In: *CoRR* abs/1703.08109 (2017).

[28] László Babai. *Handbook of Combinatorics (Vol. 2)*. Ed. by R. L. Graham, M. Grötschel, and L. Lovász. Cambridge, MA, USA: MIT Press, 1995. Chap. Automorphism Groups, Isomorphism, Reconstruction, pp. 1447–1540. ISBN: 0-262-07171-1.

[29] B. Alspach. "Cayley graphs with optimal fault tolerance." In: *IEEE Transactions on Computers* 41.10 (1992), pp. 1337–1339. ISSN: 0018-9340. DOI: 10.1109/12.166612.

[30] C. Camarero, C. Martínez, E. Vallejo, and R. Beivide. "Projective Networks: Topologies for Large Parallel Computer Systems." In: *IEEE Transactions on Parallel and Distributed Systems* 28.7 (2017), pp. 2003–2016. ISSN: 1045-9219. DOI: 10.1109/TPDS.2016.2635640.

[31] F. J. Andújar-Muñoz, J. A. Villar-Ortiz, J. L. Sánchez, F. J. Alfaro, and J. Duato. "N-Dimensional Twin Torus Topology." In: *IEEE Transactions on Computers* 64.10 (2015), pp. 2847–2861. ISSN: 0018-9340.

[32] David Coudert and Guillaume Ducoffe. "Data center interconnection networks are not hyperbolic." In: *Theoretical Computer Science* 639 (2016), pp. 72 –90. ISSN: 0304-3975. DOI: http://dx.doi.org/10.1016/j.tcs.2016.05.025.

[33] Paul R. Hafner. "Large Cayley Graphs and Digraphs with Small Degree and Diameter." In: *Computational Algebra and Number Theory*. Ed. by Wieb Bosma and Alf van der Poorten. Dordrecht: Springer Netherlands, 1995, pp. 291–302. ISBN: 978-94-017-1108-1. DOI: 10.1007/978-94-017-1108-1_21.

[34] M. Abas. "Cayley graphs of diameter two in interconnection networks." In: *Proceedings of 15th International Conference MECHATRONIKA*. 2012, pp. 1–3.

[35] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN: 0321455363.

[36] Michael Sipser. *Introduction to the Theory of Computation*. 1st. International Thomson Publishing, 1996. ISBN: 053494728X.

[37] Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of computational group theory*. Discrete mathematics and its applications. Boca Raton: Chapman & Hall/CRC, 2005. ISBN: 1-584-88372-3.

[38] D.E. Knuth and P.B. Bendix. "Simple Word Problems in Universal Algebras." English. In: *Automation of Reasoning*. Ed. by J. Siekmann and G. Wrightson. Symbolic Computation. Springer Berlin, 1983, pp. 342–376. ISBN: 978-3-642-81957-5.

[39] Derek F. Holt. *KBMAG Package: A Knuth-Bendix on Monoids, and Automatic Groups*. https://www.gap-system.org/Packages/kbmag.html. Accessed: 2018-02-23. 2017.

[40] M. Gromov. "Hyperbolic groups." In: *Essays in group theory*. Vol. 8. Math. Sci. Res. Inst. Publ. Springer, New York, 1987, pp. 75–263. DOI: 10.1007/978-1-4613-9586-7_3.

[41] M. Tokuda, Y. Hirai, and K. Kaneko. "An Algorithm for k-Pairwise Cluster-Fault-Tolerant Disjoint Paths in a Burnt Pancake Graph." In: *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2015, pp. 651–655. DOI: 10.1109/CSCI.2015.117.

[42] CHARLES C. SIMS. "Computational methods in the study of permutation groups." In: *Computational Problems in Abstract Algebra*. Elsevier, 1970, pp. 169–183. DOI: 10.1016/b978-0-08-012975-4.50020-5.

[43] Junghun Ryu, Jaewook Yu, Eric Noel, and K. Wendy Tang. "Borel Cayley Graph-Based Topology Control for Consensus Protocol in Wireless Sensor Networks." In: *ISRN Sensor Networks* 2013 (2013), p. 15. DOI: http://dx.doi.org/10.1155/2013/805635.

[44] Dongsoo Kim, Eric Noel, and K. Wendy Tang. "Expanded Borel Cayley Graphs (Ex-BCGs): A novel communication topology for multi-agent systems." In: *Journal of Network and Computer Applications* 37 (2014), pp. 47 –61. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2012.12.014.

[45] J. Ryu, E. Noel, and K. W. Tang. "Fault-tolerant routing on Borel Cayley graph." In: *IEEE International Conference on Communications (ICC)*. June 2012, pp. 2872–2877. DOI: 10.1109/ICC.2012.6364037.