Multi-objective Optimization in Graphical Models

Emma Rollón

Advisor: Javier Larrosa

Universitat Politècnica de Catalunya
Departament de Llenguatges i sistemes informàtics
Barcelona, Spain

A mis padres

A todos los que han compartido conmigo este camino,
apoyándome y acompañándome

Contents

Acknowledgements

First of all, I would like to express my utmost gratitude to my supervisor Javier Larrosa for his invaluable advice and patient guidance throughout this *journey*. I deeply thank you for keeping me focused and cheer me up in every single moment. Actually, you made this Thesis real.

I'm very thankful to my *pals*. With them I have lived terrific moments that have given me strenght to every day life. Thanks to Sara, Jordi, Wal (my always on-line personal English translator), Juan ... and all the others to being always there.

I also want to thank my basketball mates. They have given a new dimension to the sentence *mens sana in corpore sano*. In particular, I thank Roser for those discussions that, very often, only the two of us comprenhend; and Txus, for exactly the opposite.

I'm full of gratitude to my *suisse* friends. Thank you for letting me feel like home, for your support and help. Specially, I want to thank my confidant Guillermo for all we have lived together.

I thank my office mates at room C6 for the friendship that started there, and will continue wherever we go.

I want to acknowledge Stefano Bistarelli and Fabio Gaducci for our discussions on Chapter 4.

I also would like to thank all my research colleagues, with whom I have shared many conferences and seminars. With them, all these meetings have looked like hollidays rather than work.

Finally, special thanks to my family for giving me support and care.

Abstract

Many real-life optimization problems are *combinatorial*, *i.e.* they concern a choice of the best solution from a finite but exponentially large set of alternatives. Besides, the solution quality of many of these problems can often be evaluated from several points of view (a.k.a. *criteria*). In that case, each criterion may be described by a different *objective function*. Some important and well-known multicriteria scenarios are:

- In investment optimization one wants to minimize risk and maximize benefits.
- In travel scheduling one wants to minimize time and cost.
- In circuit design one wants to minimize circuit area, energy consumption and maximize speed.
- In knapsack problems one wants to minimize load weight and/or volume and maximize its economical value.

The previous examples illustrate that, in many cases, these multiple criteria are *incommensurate* (i.e., it is difficult or impossible to combine them into a single criterion) and *conflicting* (i.e., solutions that are good with respect one criterion are likely to be bad with respect to another). Taking into account simultaneously the different criteria is not trivial and several notions of optimality have been proposed. Independently of the chosen notion of optimality, computing optimal solutions represents an important current research challenge [41].

Graphical models are a *knowledge representation* tool widely used in the Artificial Intelligence field. They seem to be specially suitable for combinatorial problems. Roughly, graphical models are graphs in which nodes represent *variables* and the (lack of) arcs represent conditional independence assumptions. In addition to the graph structure, it is necessary to specify its *micro-structure* which tells how particular combinations of instantiations of interdependent variables interact. The graphical model framework provides a unifying way to model a broad spectrum of systems and a collection of general algorithms to efficiently solve them.

In this Thesis *we integrate multi-objective optimization problems into the graphical model paradigm* and study how algorithmic techniques developed in the graphical model context can be extended to multi-objective optimization problems. As we show, multi-objective optimization problems can be formalized as a particular case of graphical models using the semiring-based framework [17]. It is, to the best of our knowledge, the first time that graphical models in general, and semiring-based problems in particular are used to model an optimization problem in which the objective function is partially ordered. Moreover, we show that most of the solving techniques for mono-objective optimization problems can be naturally extended to the multi-objective context. The result of our work is the mathematical formalization of multi-objective optimization problems and the development of a set of multiobjective solving algorithms that have been proved to be efficient in a number of benchmarks.

Chapter 1

Introduction

1.1 Motivation

Graphical models [35, 111, 148, 71, 29] provide a common formalism to describe a wide range of systems. Graphical models have been adopted in a wide variety of application areas, including *genetics* [46, 93], *error-correcting codes* [102], *speech processing* [116], *image analysis* [49, 9, 142], *computational biology* [130, 117], *scheduling* [12] and *electronic commerce* [78].

A graphical model consists on a set of *variables*, a finite set of *domain values*, and a set of *functions*. The *variables* represent the objects or items that can undertake different domain values. The set of possible domain values that each variable can take is its *domain*. Finally, the set of *functions* associate *valuations* to the different possible variable assignments. The valuation tells how good or bad the assignment is from a local perspective. Valuations may represent *preferences*, *priorities*, *costs* or *probabilities* among assignments. Different instantiations of the graphical model framework differ in the meaning of the valuations, the way valuations are combined in order to get a global view, and the type of queries asked to the model [17].

Many important formalisms fall into the category of graphical models. The conceptually simplest case is *constraint networks* [96] which formalize real world deterministic problems, such as *scheduling*, *timetabling*, *proposi-*

tional reasoning, etc. Functions in constraint networks are called *constraints* and return a boolean value. Value *true* means that the assignment is permitted and *false* means that it is forbidden. In other words, constraints impose limitations on the domain values that a given set of variables can take. Solutions are assignments of domain values to variables respecting the problem constraints. The common task over constraint networks, called constraint satisfaction problem (CSP), is to find a solution, or to prove that there is none. Many research has been devoted to constraint satisfaction in the last three decades. In particular, *Constraint Programming* is a research field whose main goal is the development of languages and algorithms to model and solve problems that can be expressed as CSPs [35, 126].

Another important instantiation of the graphical model framework are *cost networks* [35]. Functions in cost networks return a number which indicates how good a partial assignment is. They are able to model many real world mono-objective *optimization* problems. The most common task over cost networks, called weighted constraint satisfaction problem (WCSP), is to find an *optimal* (i.e., maximal or minimal) solution.

Although mono-objective optimization is an extremely important class of problems, many important real world optimization problems are *multi-objective*. They involve multiple, conflicting, and non commensurate objectives. The simultaneous optimization of different measures differ with the single measure optimization in that there does not exist an unique perfect solution, but a set of incomparable ones. Multi-objective optimization is a well-established research field in Mathematical Programming. However, we observed that very little work has been done in the context of Artificial Intelligence. The repeated identification of multi-objective problems and the lack of specific solving techniques from Artificial Intelligence in general and graphical models in particular, led us to explore multi-objective optimization problems in the graphical model context. To the best of our knowledge, this is a novel approach.

The common view given by the graphical model framework has many

advantages. In particular, specialized techniques that have been developed for one type of graphical model can be transferred to another. This Thesis is built upon previous work on cost networks. Our work is concerned with the development of efficient algorithms for multi-objective optimization. The central idea of this Thesis is to model multi-objective optimization tasks within the graphical model framework and extend previous mono-objective techniques to the multi-objective case. Our study takes into account the two main solving techniques: *systematic search* and *complete inference*. In addition, we consider *lower bounding* methods that compute quasi-optimal solutions. Such methods play a fundamental role in combination with search because they can be used to prune regions of the search space.

1.2 Scope and Orientation

The boundaries of this work are established by the following decision:

- *Additive objective functions.* With the exception of Chapter 4, we restrict our work to problems with all objective functions being additive. An objective function is additive if it has the form $F(\mathcal{X}) = \sum_i f_i(\mathcal{X}_i)$, where $\mathcal{X}$ is the set of variables and $\mathcal{X}_i \subseteq \mathcal{X}$ is the scope of function f_i. It is important to remark that this type of objective functions are very general and include many significant applications. Moreover, it is worth noting that many ideas developed for additive objective functions can be directly used in other types of objective functions (e.g., $F(\mathcal{X}) = \max_i\{f_i(\mathcal{X}_i)\}$ or $F(\mathcal{X}) = \prod_i f_i(\mathcal{X}_i)$).

- *Optimization as minimization.* We assume optimization as minimization. This decision is done without loss of generality, as the maximization of any measure can be expressed as the minimization of its complementary.

- *Pareto optimality.* The simultaneous optimization of multiple objectives deviates from single objective optimization in that it does not

admit a single, optimal solution. The study of notions of optimality that are both simple and practical is very important. In this work, we adopt the very general notion of pareto-optimality which is characterized by a family of solutions which must be considered equivalent in the absence of information concerning the relevance of each objective relative to the others. These solutions are optimal in the wider sense that no other solutions in the space of possibilities are superior to them when *all* objectives are considered. It is important to note that although the number of pareto optimal solutions can be exponentially large, in practice many problems have a relatively small number of pareto-optimal solutions.

- *General-purpose algorithms.* In our work we only develop general purpose techniques. We do not make any assumption about the problems that we attempt to solve. In practice, it means that our algorithms takes the problem in its implicit way, and cannot take advantage of its peculiarities. For this reason, our methods are expected to be applicable to a broad spectrum of domains. It is clear that dedicated algorithms may perform better in their specific domains. However, general algorithms are a reasonable first step toward more efficient specialized algorithms.

- *Local search methods.* In our work we disregard local search methods. They are approximation methods based on search. Local search methods have recently become very popular because they work extremely well in some problem. In particular, *genetic algorithms* have been widely studied in the multi-objective optimization context. We restrict ourselves to the less studied exact methods.

- *Empirical evaluation.* Most of the algorithms that we present have an exponential worst-case behaviour. However, it is well known that some particular instances may be *easy* for these algorithms. Therefore, we assess the efficiency of each new algorithm empirically. We run all

the experiments in a Pentium IV at 3 GHz. with 2 Gb. of memory, over linux. The final objective of our work is to contribute to the development of algorithms that can actually be applied in real domains. In that sense, every idea that we explore has immediate algorithmic implications that we motivate and develop. Therefore, the end-product of our contributions are specific algorithms that have been implemented and can be tested on any graphical model problem.

We also want to say upfront that it is not our goal to outperform *Mathematical Programming* techniques. It would be unrealistic and way too ambitious to expect to improve several decades of previous research. Our only goal is to provide the foundations of a fresh point of view to multi-objective optimization.

- *Benchmarks.* In our empirical evaluation, we use an heterogeneous set of benchmarks composed by both academic (i.e., random) and real-world inspired problems. The ciclicity of the graph representation of these instances covers all degrees, from very low (almost a tree) to very high (almost a clique). As we will see, the degree of ciclicity of each instance will determine the *a priori* more suitable algorithm for solving it. Finally, all our benchmarks are bi-objective. This decision is made without loss of generality and for simplicity reasons. It is important to note that, in general, real-world problems will not have more than three or four objectives.

1.3 Contributions

1.3.1 Algebraic Formalization of Multi-objective Problems

The Semiring CSP (SCSP) framework [18] axiomatically describes reasoning tasks on graphical models. Its main goal is to capture many optimization

problems in a common formal framework. Then, algorithmic techniques can also be described within the SCSP framework. As a result, any new problem expressed as a SCSP immediately inherits the techniques developed for the framework.

It is well-known that SCSPs capture the most important mono-objective problems such as CSP, *fuzzy* CSP, *weighted* CSP, etc. Moreover, it has been claimed that the SCSP framework is able to model problems where the measure to be optimized is partially ordered. Arguably, the most frequent case of such problems is multi-objective optimization. Up to now, there is no completely satisfactory formalization of pareto-optimality optimization within the SCSP framework.

Our first contribution is the formalization of multi-objective problems as particular instances of the SCSP framework. In particular, we show how to build from a partially ordered semiring $\mathcal{K}$ a new semiring $\mathcal{L}(\mathcal{K})$ such that the result of the corresponding optimization task is the *set* of incomparable costs associated with each optimal assignment in $\mathcal{K}$. This formalization is pivotal to our work and it is used through the rest of the dissertation.

1.3.2 Multi-objective Branch-and-Bound Search

Depth-First Branch-and-Bound (DF-BB) [106] is a well-known systematic search algorithm widely used for mono-objective optimization tasks. It traverses depth-first a tree where each node is associated with a partial assignment. In each step, the algorithm *guesses* which is the next variable and the next domain value to assign. If the guess does not succeed, the algorithm undoes the assignment (i.e., prunes the current subtree) and tries a different one. Else, it proceeds recursively. During the search, the algorithm keeps track of the best solution found so far, which is an *upper bound* of the optimal solution. In each node, DF-BB computes a *lower bound* of the best-cost solution that can extend the current assignment. When the current lower bound is greater or equal to the upper bound the algorithm backtracks to a previous node, because the current path cannot lead to a better solution

than the current best one.

Our second contribution is the extension of DF-BB to multi-objective optimization problems. The resulting algorithm is called *multi-objective branch-and-bound* (MO-BB). The relevance of this contribution is two-fold:

- We formally define the concepts of *lower* and *upper bound frontiers*, which are the multi-objective extensions of the lower and upper bounds used in DF-BB, and redefine the pruning condition in terms of these multi-objective bounds.
- We empirically demonstrate that MO-BB is an efficient solving technique for some multi-objective problems.

1.3.3 Multi-objective Russian Doll Search

Russian Doll Search (RDS) [145] is a well-known mono-objective depth-first branch-and-bound search algorithm which invests in high quality bounds. The idea of RDS is to replace one search by n successive searches on nested subproblems, where n is the number of variables in the problem. The key of the algorithm is that the optimal cost of each solved subproblem is systematically recorded in order to help future searches.

Our third contribution is the extension of RDS from mono-objective to multi-objective optimization. The resulting algorithm is called *Multi-objective Russian Doll Search* (MO-RDS). MO-RDS is an interesting refinement of MO-BB that appears to be efficient for solving problems with relatively small bandwidth. In particular, MO-RDS solves for the first time an open instance from the Spot5 benchmark.

1.3.4 Multi-objective Bucket Elimination

Bucket Elimination [34, 15] (BE) is one of the most significant inference algorithms for graphical models. BE eliminates variables one by one, while deducing the effect of the eliminated variables on the rest of the problem.

The elimination of the last variable produces a constant which is the optimal cost of the problem.

Our fourth contribution is the extension of BE to multi-objective optimization problems. The resulting algorithm is called *multiobjective bucket elimination* (MO-BE). We prove the theoretical complexity of the algorithm which clearly indicates the suitability of MO-BE for problems with small induced width. We demonstrate empirically the efficiency of the algorithm on those problems.

1.3.5 Multi-objective Mini-Bucket Elimination

Algorithms that compute lower bounds are a fundamental component of branch-and-bound because they can be executed at every branch node in order to detect and prune redundant subtrees. Moreover, they can be used to approximate the solution of a difficult problem that cannot be solved exactly.

Many lower bounding algorithms have been proposed in the mono-objective context. In particular, *mini-bucket elimination* (MBE) [38], the approximation version of BE, is a powerful mechanism for lower bound computation.

Our fifth contribution is the extension of MBE to multi-objective optimization problems. The resulting algorithm, called *multi-objective mini-bucket elimination* (MO-MBE), can be used to compute multi-objective lower bounds of different accuracies. The relevance of this contribution is threefold:

- We address the lack of general approximation algorithms that prevents multi-objective branch-and-bound from being widely used [41].
- We embed MO-MBE in MO-BB, and empirically demonstrate the performance of the new search approach.
- We demonstrate the accuracy of MO-MBE when used as an standalone approximation algorithm.

1.3.6 Multi-objective Constraint Propagation

In the context of constraint programming, *propagation* is the process of detecting whether a partial assignment cannot be extended to a solution. Typically, constraint propagation takes place after each assignment in a search algorithm. Most propagation algorithms detect and discard domain values that are inconsistent with the current assignment. If some variable loses all its values, the algorithm backtracks. Typically, each constraint is propagated independently. Namely, a domain value is removed if it is shown to be inconsistent with respect to one of the constraints. The only *communication* between constraints is through *domain value pruning* (pruning one domain value due to one constraint, may produce the pruning of another domain value due to another constraint, yielding a cascade effect). This solving approach may not be strong enough for problems with several conflicting constraints of the form $\sum_{f \in \mathcal{F}_j} f < K_j$.

Our sixth contribution is a novel propagation schema. Roughly, we propose to jointly propagate these constraints using multi-objective approximation algorithms. We demonstrate empirically the efficiency of this approach.

1.3.7 Engineering Mini-bucket Elimination

As we have seen, mini-bucket elimination (MBE) is one of the most popular bounding techniques for mono-objective optimization problems [38, 74, 109]. However, the time and space complexity of MBE is exponential in a control parameter z. It is important to note that, with current computers, it is the space, rather than the time, that prohibits the execution of the algorithm beyond certain values of z.

Our seventh contribution is the development of two methods to improve the practical applicability of MBE. Given a value for the control parameter z,

- the first approach decreases its space demands and obtains the same lower bound as the original MBE; and

- the second one increases its lower bound and maintains the same space demands as the original MBE.

For simplicity reasons, this contribution have been developed in the context of mono-objective optimization. However, their multi-objective extension is direct.

1.4 Publications

[123] "Multi-Objective Russian Doll Search". Emma Rollon and Javier Larrosa. *Proc. of the 22th AAAI Conference on Artificial Intelligence*, AAAI 2007.

[120] "Bucket Elimination for Multiobjective Optimization Problems". Emma Rollon and Javier Larrosa. *Journal of Heuristics*. Vol. 12, Number 4-5, pp. 307-328. September 2006.

[124] "Constraint Optimization Techniques for Multi-Objective Branch-and-Bound Search". Emma Rollon and Javier Larrosa. *Lecture Notes in Economics and Mathematical Systems*, In Press.

[121] "Mini-bucket Elimination with Bucket Propagation". Emma Rollon and Javier Larrosa. *Proc. of the 12th International Conference on Principles and Practice of Constraint Programming*, LNCS 4204, pp. 484-498, CP 2006.

[122] "Multi-Objective Propagation in Constraint Programming". Emma Rollon and Javier Larrosa. *Proc. of the 17th European Conference on Artificial Intelligence*, ECAI 2006.

[119] "Depth-First Mini-bucket Elimination". Emma Rollon and Javier Larrosa. *Proc. of the 11th International Conference on Principles and Practice of Constraint Programming*. LNCS 3709, pp. 563-577, CP 2005.

[92] "Bucket Elimination with Capacity Constraints". Javier Larrosa and Emma Rollon. *6^{th} Workshop on Preferences and Soft Constraints*. Attached to the 10th International Conference on Principles and Practice of Constraint Programming 2004, CP 2004.

[91] "Adaptive Consistency with Capacity Constraints". Javier Larrosa and Emma Rollon. *Workshop on Modeling and Solving Problems with Constraints*. Attached to the 16^{th} European Conference on Artificial Intelligence. ECAI 2004.

1.5 Overview

This Thesis is structured in ten Chapters and two Appendices. Chapter 2 introduces and motivates the main elements that will be used throughout our work: *multi-objective optimization* and *graphical models*. The first one is the type of problems we consider and the second one is the main tool we use to solve them. Moreover, it presents some basic notation used in this Thesis.

Chapter 3 revises previous work on graphical models and multi-objective optimization. Regarding graphical models, we present the most important general algorithms for mono-objective optimization. It covers search and inference algorithms as the main exact solving methods, along with local consistency and mini-bucket elimination algorithms as the main approximation methods. Regarding multi-objective optimization, we overview relevant algorithms developed in the Artificial Intelligence and Operational Research areas.

Chapter 4 describes multi-objective optimization problems in terms of graphical models. We develop this formalization inside of the *Semiring CSP* framework, a well-known algebraic framework to axiomatize graphical models. Moreover, we show that previous attempts to describe multi-objective optimization as a particular graphical model do not capture the notion of pareto optimality. Finally, we specialize the formalization to the case of minimizing additive objective functions, which are the type of problems selected

for our algorithmic study.

Chapter 5 extends branch-and-bound (BB) to multi-objective optimization. The resulting algorithm is called multi-objective branch-and-bound (MO-BB). Moreover, we present some direct extensions of the classic mono-objective lower bounds to the multi-objective case. The experimental results show that these multi-objective lower bounds must be the starting point to more sophisticated ones.

Chapter 6 extends russian doll search (RDS) to multi-objective optimization. More precisely, we extend the standard and specialized versions of russian doll search. The resulting algorithms are called multi-objective russian doll search (MO-RDS) and specialized multi-objective russian doll search (SMO-RDS). The key point in these extensions is the use of the previous MO-BB algorithm along with more sophisticated multi-objective lower bounds. Moreover, we empirically demonstrate that sometimes it may be convenient to solve mono-objective optimization problems as if they were multi-objective. Specifically, this transformation allows us to solve for the first time an open instance of the Spot5 benchmark using MO-RDS.

Chapter 7 extends bucket elimination (BE) to multi-objective optimization. The resulting algorithm is called multi-objective bucket elimination (MO-BE). We prove that its complexity is exponential in an structural parameter called induced width. This complexity renders MO-BE feasible for problems with small induced width. We assess the suitability of MO-BE on bi-objective optimization random and real-world inspired problems.

Chapter 8 extends mini-bucket elimination (MBE), a well-known mono-objective approximation algorithm, to multi-objective optimization. The resulting algorithm is called multi-objective mini-bucket elimination (MO-MBE). It computes multi-objective lower bounds. MO-MBE can be used as an stand-alone algorithm to compute an approximation of the pareto optimal solutions or as a bounding evaluation function inside multi-objective branch-and-bound. We assess the suitability of the new algorithm in both cases on bi-objective random and real-world inspired problems.

Chapter 9 proposes a new method to simultaneously propagate a set of constraints while solving constraint satisfaction problems during search. After presenting how CSP solvers propagate each constraint independently, we propose a more convenient approach. Essentially, the approach consists in considering CSP problems as multi-objective minimization problems. Then, we compute a multi-objective lower bound using multi-objective mini-bucket elimination that, if large enough, allows backtracking. We demonstrate the suitability of this approach on two domains inspired on real-world problems.

Chapter 10 gives the conclusions of our work and proposes some lines of future work.

Appendix A describes two methods to improve the practical applicability of MBE. For simplicity reasons these two methods are proposed for mono-objective optimization. However, they can be also extended to the multi-objective case. In the first part of the Appendix, we introduce a method to reduce the space demands while obtaining the same lower bound as the original MBE algorithm. In the second part of the Appendix, we introduce a method to increase the lower bound while maintaining the same space demands as the original MBE algorithm. We assess the improvements of both approaches empirically on mono-objective random and real-world inspired problems.

Appendix B describes the benchmarks used throughout our work. For each benchmark, we describe the problems, the instances included and how they have been generated, and how they are encoded in the graphical model framework. Moreover, we indicate their important structural properties as the induced width and the graph bandwidth.

Chapter 2

Preliminaries

The purpose of this chapter is to introduce the two main elements that will be used throughout this thesis: *multi-objective optimization* and *graphical models*. Recall that multi-objective problems are the type of problems that we address and graphical models are the conceptual tool that we use to view and efficiently solve them.

After presenting basic notation on tuples and functions (Section 2.1), we introduce the main notions around multi-objective optimization and the type of problems that we consider (Section 2.2). Afterward, we overview the notion of graphical model (Section 2.3) which constitutes the central tool of our work. We show the expressive power of the graphical framework by presenting its two most prominent instantiations (i.e., constraint networks and bayesian networks) and extensions. Finally, we review some important graph concepts and show their connection with the complexity of solving tasks posed to graphical models (Section 2.4).

2.1 Basic Notation

The type of problems addressed in this Thesis are defined in terms of a set of *variables* taking values on finite sets of *domain values* and a set of *functions* defined over these variables. Roughly, solving a problem is somehow related

to assigning domain values to the variables and evaluating the functions in those assignments.

In the following, $\mathcal{X} = (x_1, \ldots, x_n)$ is an ordered set of variables and $\mathcal{D} = (D_1, \ldots, D_n)$ is and ordered set of domains, where D_i is the finite set of potential domain values for x_i.

2.1.1 Tuples

The *assignment* (i.e, instantiation) of variable x_i with domain value $a \in D_i$ is noted $(x_i = a)$. A *tuple* is an ordered set of assignments to different variables $(x_i = a_i, \ldots, x_j = a_j)$. The *scope* of tuple t, noted $var(t)$, is the set of variables that it assigns. When the scope of a tuple is clear by the context we only write its domain values.

We need two basic operations over tuples: *join* and *projection*. Let t and t' be two tuples such that both assign the same value to their common variables. Their *join*, noted $t \cdot t'$, is a new tuple which contains the assignments of both t and t'. The *projection* of t over $\mathcal{Y} \subseteq var(t)$, noted $t[\mathcal{Y}]$, is a sub-tuple of t containing only the instantiation of variables in $\mathcal{Y}$. Projecting a tuple t over the empty set $t[\emptyset]$ produces the empty tuple, noted λ. A *complete assignment* is an assignment of all the variables in $\mathcal{X}$ and we will usually refer to it as X.

Example 2.1.1 *Consider three variables $\mathcal{X} = \{x_1, x_2, x_3\}$ taking domain values over a common domain $D_1 = D_2 = D_3 = \{a, b, c\}$. Consider three tuples $t = (x_1 = a, x_3 = b)$, $t' = (x_1 = a, x_2 = c)$ and $t'' = (x_1 = b, x_2 = c)$. The join of t and t' is $t \cdot t' = (x_1 = a, x_2 = c, x_3 = b)$. The join of t and t' is not applicable because they assign a different value to the common variable x_1. The projection of t over variable x_1 is $t[x_1] = (x_1 = a)$.*

2.1.2 Functions

For a subset of variables $\mathcal{Y} \subseteq \mathcal{X}$, we note $l(\mathcal{Y})$ the set of tuples over $\mathcal{Y}$. Let $f : l(\mathcal{Y}) \to A$ be a function defined over $\mathcal{Y}$. The *scope* of f, noted $var(f), is\mathcal{Y}$.

The *arity* of f, noted $|var(f)|$, is the size of its scope. The set A is called the set of *valuations* and it is problem specific. The *instantiation* of function f by tuple t, noted $f(t)$, is a new function in which variables in $var(f) \cap var(t)$ are *fixed* as indicated by t. The scope of $f(t)$ is $var(f) - var(t)$[1]. When the scope of $f(t)$ is not empty, the function f has been *partially* instantiated. Otherwise, the function f has been *totally* instantiated and $f(t)$ is a singleton of A.

Example 2.1.2 *Consider a function $f(x_1, x_2) = 3x_1x_2$ where x_1 and x_2 take values on some interval of the naturals. The scope of f is $var(f) = \{x_1, x_2\}$. The set of valuations is $\mathbb{N}$. Let $t = (x_1 = 1, x_3 = 2)$ be a tuple. The instantiation of f by t (i.e., $f(t)$) is a new function $g(x_2) = 3x_2$. Note that, since $var(t) \not\subseteq var(f)$, just the assignment of x_1, which is the common variable between t and f, is relevant in $f(t)$. Moreover, the instantiation is partial, because the scope of the new function is not empty. Let $t' = (x_2 = 3)$ be a new tuple. The instantiation $g(t')$ is a new function $g'() = 9$. Since all the variables in $var(g)$ has been instantiated by t', $g(t')$ is totally instantiated.*

In the previous example both variables and functions were numeric (over the naturals). However, it is important to note that in general, variables can take domain values on arbitrary finite domains, and functions can return values from an arbitrary set of valuations.

There are two operations over valuations: *combination* and *addition*.

Definition 2.1.1 *A* valuation structure *is a triple $\mathcal{K} = (A, \otimes, \oplus)$ such that A is an arbitrary set of valuations, $\otimes$ is a binary operation $\otimes : A \times A \to A$ called* combination *and $\oplus$ is a binary operation $\oplus : A \times A \to A$ called* addition*. Both operators are associative and commutative.*

In Chapter 4 we discuss in detail the algebraic properties that valuation structures must satisfy in the graphical models context.

[1]We denote by $\mathcal{Z} - \mathcal{W}$ the usual set difference defined as $\mathcal{Z} - \mathcal{W} = \{a \in \mathcal{Z} \mid a \notin \mathcal{W}\}$

Abusing notation we extend the combination and addition operators to operate also over functions. The *combination* of two functions f and g is a new function $f \bigotimes g$ that *aggregates* their information.

Definition 2.1.2 *Let $f : l(var(f)) \rightarrow A$ and $g : l(var(g)) \rightarrow A$ be two functions. Their* combination*, noted $f \bigotimes g$, is a new function h with scope $var(f) \cup var(g)$ which returns for each tuple $t \in l(var(f) \cup var(g))$ the combination of valuations of f and g. Formally,*

$$\forall\, t \in l(var(f) \cup var(g)),\ h(t) = f(t) \otimes g(t)$$

Example 2.1.3 *Typical combination operators are logical and (i.e., $\wedge$) over booleans and sum (i.e., $+$) over numbers. For instance, if x_1, x_2 and x_3 are boolean variables, $f_1(x_1, x_2) = x_1 \vee x_2$ and $f_2(x_2, x_3) = \neg x_2 \vee x_3$ then $(f_1 \wedge f_2)(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$. If x_1, x_2 and x_3 are numerical variables, $f_1(x_1, x_2) = x_1 x_2$ and $f_2(x_2, x_3) = 2x_2 + x_3$ then $(f_1 + f_2)(x_1, x_2, x_3) = x_1 x_2 + 2x_2 + x_3$).*

The *marginalization* of a subset of variables $\mathcal{W} \subseteq \mathcal{X}$ from a function f is a new function $\bigoplus_{\mathcal{W}} f$ that joins/chooses information among the different possible alternatives for the variables in $\mathcal{W}$. Specifically, $\bigoplus_{\mathcal{W}} f$ removes from the scope of f the variables in $\mathcal{W}$, while summarizing via addition $\oplus$ the eliminated information. Therefore, the marginalization operator narrows the focus of the valuations of f to the remaining variables.

Definition 2.1.3 *Let $f : l(var(f)) \rightarrow A$ be a function and $\mathcal{W} \subseteq \mathcal{X}$ be a set of variables. The* marginalization *of $\mathcal{W}$ from f, noted $\bigoplus_{\mathcal{W}} f$, is a new function h with scope $var(f) - \mathcal{W}$ that returns for each tuple $t \in l(var(f) - \mathcal{W})$ the addition of the valuations over the different extensions to $\mathcal{W}$. Formally,*

$$\forall\, t \in l(var(f) - \mathcal{W}),\ h(t) = \bigoplus_{t' \in l(\mathcal{W})} f(t \cdot t')$$

Sometimes, the marginalization operator is also called *elimination* operator because it removes $\mathcal{W}$ from the scope of f.

Example 2.1.4 *Typical marginalization operators are the logical or (i.e.,* $\vee$*), the minimum (i.e.,* min*), or the sum (i.e.,* $+$*). For instance, if* x_1 *and* x_2 *are boolean variables,* $f(x_1, x_2) = x_1 \vee x_2$ *and the marginalization operator is the logical or, then* $\bigoplus_{x_2} f = \vee_{x_2 \in \{true, false\}} \{x_1 \vee x_2\} = (x_1 \vee true) \vee (x_2 \vee false) = true$*. If* x_1 *and* x_2 *are numerical variables in the interval* $[1, \ldots, 5]$*,* $f(x_1, x_2) = x_1 x_2$ *and the marginalization operator is* min*, then* $\bigoplus_{x_2} f = \min_{x_2 \in [1,\ldots,5]} \{x_1 x_2\} = \min\{1x_1, 2x_1, 3x_1, 4x_1, 5x_1\} = x_1$.

2.2 Multi-objective Optimization

A (mono-objective) optimization problem is the problem of finding the *best* solution according to some criterion expressed by means of a function F. For the purposes of this Thesis, an optimization problem P is defined by a set of variables $\mathcal{X} = \{x_1, \ldots, x_n\}$ restricted to finite sets of domain values $\mathcal{D} = \{D_1, \ldots, D_n\}$ and an objective function $F : l(\mathcal{X}) \rightarrow A$, where A is a totally ordered set (usually a number). The task is to find a complete assignment X such that $\forall\, X', F(X) \leq F(X')$ (i.e., minimization).

Multi-objective optimization problems deal with multiple objectives, which should be simultaneously optimized [143, 41, 69]. Consider a problem P_{mo} with p objective functions $F_1, \ldots, F_p$. Given a complete assignment X, the problem associates a cost $F_j(X)$ to each objective j. These $F_j(X)$ costs can be represented as a vector $F(X) = (F_1(X), \ldots, F_p(X))$. Given two complete assignments X and X', their associated cost vectors can be compared in order to decide which one is better.

Definition 2.2.1 $F(X)$ dominates $F(X')$*, noted* $F(X) < F(X')$*, iff they are different and* $F_j(X)$ *is* better *than* $F_j(X')$ *for all the objectives. Formally,*

$$F(X) < F(X') \text{ iff } F(X) \neq F(X') \text{ and } \forall\, 1 \leq j \leq p,\ F_j(X) \leq F_j(X')$$

We say that $F(X) \leq F(X')$ iff $F(X) < F(X')$ or $F(X) = F(X')$. We say that $F(X)$ and $F(X')$ are *incomparable* iff $F(X) \not\leq F(X')$ and $F(X') \not\leq F(X)$. Observe that the domination relation is a partial order among vectors.

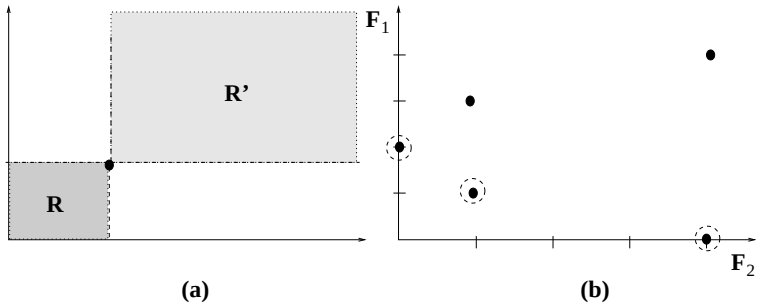

Figure 2.1: A set of cost vectors.

Example 2.2.1 *Consider bi-dimensional cost vectors. Figure 2.1 (a) depicts a vector $\vec{v}$ as a dot in the 2D space. $\vec{v}$ is the top-right corner of a rectangle R. Any point inside R would dominate $\vec{v}$. $\vec{v}$ is also the bottom-left corner of an infinitely large rectangle R'. Any point inside R' would be dominated by $\vec{v}$. Moreover, any cost vector $\vec{u}$ outside R and R' is incomparable with respect $\vec{v}$.*

The domination order among vectors also defines a partial order among complete assignments. This partial order captures the notion of preference among assignments. Clearly, we prefer X over X' (i.e., X is *better* than X') if its associated cost vector $F(X)$ dominates $F(X')$.

A complete assignment X is *pareto optimal* or an *efficient* solution if there is no better complete assignment. Since domination is a partial order, there may be a set of incomparable efficient solutions. The set of vectors associated to each efficient solution is called *efficient frontier*.

Definition 2.2.2 *Let X and X' denote complete assignments. The* set of efficient or pareto optimal solutions *is $\mathcal{X}_\mathcal{E} = \{X | \forall X', F(X') \not< F(X)\}$. The* efficient frontier *is $\mathcal{E} = \{F(X) | X \in \mathcal{X}_\mathcal{E}\}$.*

A very important task of interest in a multi-objective optimization problem P_{mo} is to compute its efficient frontier $\mathcal{E}$ (and, possibly, one or all efficient solutions for each of its elements).

Example 2.2.2 *Consider a problem represented by one integer variable x_1 in the range $[-2, \ldots, 2]$, and two objective functions $F_1(x_1) = x_1 + 2$ and $F_2(x_1) = x_1^2$. Figure 2.1 (b) depicts the set of all cost vectors where efficient vectors are emphasized with dotted circles. The efficient frontier is $\mathcal{E} = \{(0, 4), (1, 1), (2, 0)\}$. The set of efficient solutions $\mathcal{X}_\mathcal{E}$ is the set $\{(x_1 = -2), (x_1 = -1), (x_1 = 0)\}$.*

2.3 Graphical Models

The *graphical model* framework provides a common formalism to model several systems such as *probabilistic models*, which includes Markov and Bayesian networks [111], and *deterministic models*, which includes constraint networks and decision diagrams [35]. In general, a graphical model is defined by a collection of functions $\mathcal{F}$ over a set of variables $\mathcal{X}$. Depending on each particular case, functions may express probabilistic, deterministic or preferential information. As we will show in the next section, the structure of graphical models is naturally captured by a graph which expresses conditional independences between variables.

Definition 2.3.1 *A graphical model is a tuple $\mathcal{M} = (\mathcal{X}, \mathcal{D}, A, \mathcal{F}, \bigotimes)$ where:*

- *$\mathcal{X} = \{x_1, \ldots, x_n\}$ is a finite set of variables,*
- *$\mathcal{D} = \{D_1, \ldots, D_n\}$ is the set of their corresponding finite domains,*
- *A is a set of valuations,*
- *$\mathcal{F} = \{f_1, \ldots, f_e\}$ is a set of discrete functions such that $var(f_k) \subseteq \mathcal{X}$ and $f_k : l(var(f_k)) \rightarrow A$,*
- *$\bigotimes$ is a combination operator over functions as defined in Definition 2.1.2.*

Each function $f_k \in \mathcal{F}$ expresses *local information* concerning the interaction of variables in its scope $var(f_k)$. The combination operator allows to *aggregate* this local information and get a wider view. The *global view* of a graphical model is represented by the combination of all its functions $\bigotimes_{k=1}^{e} f_k$. Note that the scope of the global function may be the whole set of variables. Therefore, it has an exponentially large number of entries with respect to the number of variables. As a result, it is not practical to store it explicitly or even to inspect it exhaustively. What all graphical models have in common is that the global view comes into small pieces ($f \in \mathcal{F}$), usually called *factors*, that makes it manageable. In other words, graphical models represent a system as a factorization of its global information.

In many domains of application of graphical models some variables may have a pre-assigned domain value. Therefore, the original graphical model is *conditioned* to the domain value of those variables. The result is a new graphical model where those variables have been fixed to its corresponding domain value.

Definition 2.3.2 *Consider a tuple t such that $var(t) \subseteq \mathcal{X}$ and a graphical model $\mathcal{M} = (\mathcal{X}, \mathcal{D}, A, \mathcal{F}, \bigotimes)$ where $\mathcal{F} = \{f_1, \ldots, f_e\}$. $\mathcal{M}$ conditioned to t, noted $\mathcal{M}(t) = (\mathcal{X}, \mathcal{D}, A, \mathcal{F}(t), \bigotimes)$, is a new graphical model where $\mathcal{F}(t) = \{f_1(t), \ldots, f_e(t)\}$. Namely, each function in $\mathcal{F}$ has been instantiated by t.*

Once a problem has been modeled as a graphical model, the user usually wants to ask queries on it. These queries, also called *reasoning tasks*, depend on the marginalization operator $\bigoplus$.

Definition 2.3.3 *A* reasoning task *is a tuple $\mathcal{P} = (\mathcal{X}, \mathcal{D}, A, \mathcal{F}, \bigotimes, \bigoplus)$ where $(\mathcal{X}, \mathcal{D}, A, \mathcal{F}, \bigotimes)$ is a graphical model and $\bigoplus$ is a marginalization operator over functions as defined in Definition 2.1.3. The reasoning task is the computation of $\bigoplus_{\mathcal{X}} (\bigotimes_{i=1}^{e} f_i)$.*

The instantiation of the $\bigotimes$ and $\bigoplus$ operators fixes a graphical model and its associated reasoning task, respectively. Two main families of reasoning

tasks have been exhaustively studied. They can be distinguished by looking at the particular instantiation of the $\bigoplus$ operator. If $\bigoplus$ is *min* or *max*, then the reasoning task is called *optimization* and the function $F = \bigotimes_{f \in \mathcal{F}} f$ is called the *objective function*. If $\bigoplus$ is $+$, then the reasoning task is called *counting*.

In the following, we describe two well known instantiations of the graphical model framework: constraint networks and belief networks, as well as their typical extensions and reasoning tasks. Constraint and weighted constraint networks are introduced because their multi-objective extension will be used throughout this Thesis. Belief networks are only presented to illustrate the generality of the model.

2.3.1 Constraint Networks and Extensions

Constraint networks [35] provide a framework for formulating real world deterministic problems, such as scheduling, design, planning or diagnosis. The feature that makes constraint networks unique over other graphical models is that variables take domain values on arbitrary finite domains and functions are boolean (i.e., they associate a boolean valuation to each assignment of variables). In constraint networks boolean functions are called *constraints*. The purpose of constraints is to specify that some partial assignments are forbidden (i.e., inconsistent). More precisely, given a tuple t such that $var(f) \subseteq var(t)$, if $f(t)$ is $false$ it means that constraint f forbids tuple t. If $f(t)$ is $true$ it means that tuple t satisfies constraint f. We say that t is consistent if it satisfies all its completely assigned functions. Accordingly, the combination operator in constraint networks is the logical and $\wedge$.

Definition 2.3.4 *A* constraint network *is a graphical model*

$$(\mathcal{X}, \mathcal{D}, \{true, false\}, \mathcal{C}, \wedge)$$

where $\mathcal{X}$ and $\mathcal{D}$ are variables and associated domains, respectively. $\mathcal{C} = \{f_1, \ldots, f_e\}$ is a set of boolean functions (i.e., $f_k : l(var(f_k)) \rightarrow \{true, false\}$), called constraints. The combination operator is the logical and (i.e., $\wedge$).

The usual task posed to a constraint network is called constraint satisfaction problem and consists on finding whether it is consistent or not. A constraint network is consistent if there exists a solution, that is, an assignment of all the variables, that does not violate any constraint.

Definition 2.3.5 *A* constraint satisfaction problem *(CSP) is a reasoning task* $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \{true, false\}, \mathcal{C}, \wedge, \vee)$.

Clearly, if the constraint network is consistent the result of solving the CSP is $true$. Otherwise, the constraint network is inconsistent and the result is $false$.

Example 2.3.1 *Consider a problem in which we have four objects that we must either take or leave behind and some constraints about the different object incompatibilities. We can represent this with four variables* $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$*, one for each object, and two values per domain* $D_i = \{true, false\}$ *(meaning* take *and* discard*, respectively). Object incompatibilities can be modeled as constraints between variables. Suppose that there are the following:*

- *either* x_1 *or* x_3 *should be chosen, but not both at the same time:*

$$h_1(x_1, x_3) = (x_1 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$$

- *either* x_3 *or* x_4 *must be chosen:*

$$h_2(x_3, x_4) = x_3 \vee x_4$$

- x_4 *and* x_2 *cannot be chosen at the same time:*

$$h_3(x_2, x_4) = \overline{x_4} \vee \overline{x_2}$$

The corresponding constraint network is $\mathcal{M} = (\mathcal{X}, \mathcal{D}, A, \mathcal{C}, \bigotimes)$ *where*

- $\mathcal{X} = \{x_1, \ldots, x_4\}$

- $\mathcal{D} = \{D_i = \{true, false\}\}_{i=1}^{4}$
- $A = \{true, false\}$
- $\mathcal{C} = \{h_1, h_2, h_3\}$
- $\bigotimes = \wedge$

The global view of this constraint network is a new constraint $C(\mathcal{X}) = h_1(x_1, x_3) \wedge h_2(x_3, x_4) \wedge h_3(x_2, x_4)$. *Since* C *is a boolean function, it can be represented as a truth table on all possible instantiations of its set of variables,*

x_1	x_2	x_3	x_4	
f	f	f	f	f
f	f	f	t	f
f	f	t	f	t
f	f	t	t	t
f	t	f	f	f
f	t	f	t	f
f	t	t	f	t
f	t	t	t	f
t	f	f	f	f
t	f	f	t	t
t	f	t	f	f
t	f	t	t	f
t	t	f	f	f
t	t	f	t	f
t	t	t	f	f
t	t	t	t	f

where t *and* f *are short-hands for* $true$ *and* $false$, *respectively. Consider an assignment* X *of all variables in* $\mathcal{X}$. *Since the constraints in* $C(\mathcal{X})$ *are combined via* $\wedge$, $C(X) = false$ *means that* X *does not satisfy all constraints in* $\mathcal{C}$. *Similarly,* $C(X) = true$ *means that it satisfies all the constraints in* $\mathcal{C}$.

Consider the CSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, A, \mathcal{C}, \bigotimes, \bigoplus)$ *defined over* $\mathcal{M}$*, where the marginalization operator* $\bigoplus$ *is* $\vee$*. The result of the reasoning task* $\vee_{\mathcal{X}} C$ *is true. Note that it is easy to see whether the CSP problem is satisfiable or not by just inspecting the global truth table. The only requirement for a CSP to be satisfiable is that one entry of the global view evaluates to true.*

Finally, observe that the problem has four consistent assignments: $(x_1 = f, x_2 = f, x_3 = t, x_4 = f)$*,* $(x_1 = f, x_2 = f, x_3 = t, x_4 = t)$*,* $(x_1 = f, x_2 = t, x_3 = t, x_4 = f)$ *and* $(x_1 = t, x_2 = f, x_3 = f, x_4 = t)$*.*

Observation 2.3.1 *Note that constraint networks can also be defined in terms of numbers where boolean value true is associated to* 1 *and boolean value* $false$ *is associated to* 0 *in the set of valuations* A*. Consequently, the logical and* $\wedge$ *must be replaced by the multiplication. Formally, a constraint network is a graphical model* $(\mathcal{X}, \mathcal{D}, \{1, 0\}, \mathcal{C}, \times)$ *and a CSP is a reasoning task* $(\mathcal{X}, \mathcal{D}, \{1, 0\}, \mathcal{C}, \times, \max)$*. Note that the logical or (i.e.,* $\vee$*) is replaced by* max*. Therefore, CSPs can be seen as a (degenerated) case of optimization task, with only two possible valuations* 1 *and* 0*. It is easy to see that the constraint network is consistent if the result of solving the CSP is* 1*, and inconsistent otherwise.*

Another task over a constraint network is to count the number of solutions. This reasoning task is better defined over a constraint network as stated in the previous observation.

Definition 2.3.6 *A* counting problem *is a reasoning task* $(\mathcal{X}, \mathcal{D}, \{1, 0\}, \mathcal{C}, \times, +)$*.*

Example 2.3.2 *Consider the problem from example 2.3.1 and the reasoning task of counting its solutions. Constraints are now expressed as* $1/0$ *functions,*

$$h_1(x_1, x_3) = \begin{cases} 1, & (x_1 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \\ 0, & otherwise \end{cases}$$

$$h_2(x_3, x_4) = \begin{cases} 1, & x_3 \vee x_4 \\ 0, & otherwise \end{cases}$$

$$h_3(x_2, x_4) = \begin{cases} 1, & \overline{x_2} \vee \overline{x_4} \\ 0, & otherwise \end{cases}$$

The constraint network is equivalent to that in example 2.3.1 but defined in terms of numbers. Therefore, its global view is the same constraint C expressed as a $1/0$ function (i.e., boolean valuations t and f are interpreted as natural valuations 1 and 0, respectively). Consider an assignment X of all variables in $\mathcal{X}$. Since the $1/0$ functions in C are combined via $\times$, $C(X) = 0$ means that X does not satisfy all constraints in $\mathcal{C}$. Similarly, $C(X) = 1$ means that X satisfies all the constraints in $\mathcal{C}$ (i.e., X is a consistent assignment).

The counting task corresponds to the number of 1's in the global view. It is easy to see that the result of this counting problem is 4.

Observation 2.3.2 *Note that constraint networks can also be defined as $(\mathcal{X}, \mathcal{D}, \{0, \infty\}, \mathcal{C}, +)$. Now, boolean value true is associated to 0, boolean value $false$ is associated to ∞ in the set of valuations A, and the logical* and *is replaced by the sum. Constraints are expressed as $0/\infty$ functions. If a tuple is forbidden, its associated cost is ∞. Otherwise, its associated cost is 0. Consequently, a CSP is defined as $(\mathcal{X}, \mathcal{D}, \{0, \infty\}, \mathcal{C}, +, \min)$. In this case, optimization is minimization. It is easy to see that the constraint network is consistent if the result of solving the CSP is 0, and inconsistent otherwise (i.e., the result is ∞).*

Cost networks [126] extends constraint networks in order to deal with optimization tasks (that we assume as minimization). They assume constraint networks expressed as in the previous observation and increase the number of possible valuations. In cost networks, functions specify the cost of the assignments (*i.e.*, they are cost functions). We make the usual assumption of restricting costs to the set of naturals with infinity, noted $\mathbb{N}^\infty$. The purpose of cost functions is to specify how much a partial assignment is preferred. Given a tuple t, if $f(t)$ is 0 it means that t is cost free and, therefore, is

a *perfect* assignment, that is, the specified function is satisfied and it does not change the overall level of preference for the given tuple. If $f(t)$ is ∞ it means that t is totally undesired and, therefore, is a *totally disliked* assignment, that is, the assignment t is forbidden (or inconsistent, in terms of constraint satisfaction) and it has to be avoided. In general, the lower the valuation is, the most preferred the assignment is.

Definition 2.3.7 *A* cost network *is a graphical model* $(\mathcal{X}, \mathcal{D}, \mathbb{N}^\infty, \mathcal{F}, +)$, *where* $\mathcal{X}$ *and* $\mathcal{D}$ *are variables and associated domains.* $\mathcal{F} = \{f_1, \ldots, f_e\}$ *is a set of natural valued cost functions (i.e.,* $f_k : l(var(f_k)) \rightarrow \mathbb{N}^\infty$*). The combination operator is the sum* $+$*.*

The main task posed to a cost network is to find its *optimal* cost, that is, the *best* among the costs of all complete assignments. The cost of a complete assignment X is $F(X) = \sum_{k=1}^{e} f_k(X)$. The best cost $F(X)$ is the one with minimum cost.

Definition 2.3.8 *A* weighted *CSP (WCSP) is a reasoning task*

$$(\mathcal{X}, \mathcal{D}, \mathbb{N}^\infty, \mathcal{F}, +, \min)$$

Example 2.3.3 *Consider Example 2.3.1. Suppose that discarding object* i *has an associated penalty* $p_i = i$*. Besides, objects* 2 *and* 3 *are complementary, meaning that if they are not taken together we get an additional penalty* $p_{23} = 3$*. This numerical information can be modeled as cost functions between variables. Making the most profitable selection of objects can be expressed as a minimization WCSP, where the task is to minimize the aggregated penalty of discarded objects. Constraints are expressed as* $0/\infty$ *functions,*

$$h_1(x_1, x_3) = \begin{cases} 0, & (x_1 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \\ \infty, & otherwise \end{cases}$$

$$h_2(x_3, x_4) = \begin{cases} 0, & x_3 \vee x_4 \\ \infty, & otherwise \end{cases}$$

$$h_3(x_2, x_4) = \begin{cases} 0, & \overline{x_2} \vee \overline{x_4} \\ \infty, & otherwise \end{cases}$$

Unary and binary cost functions are the following,

$$f_i(x_i) = \begin{cases} i, & x_i = 0 \\ 0, & otherwise \end{cases} \qquad f_{23}(x_2, x_3) = \begin{cases} 0, & x_2 \wedge x_3 \\ 3, & otherwise \end{cases}$$

The corresponding cost network is $\mathcal{M} = (\mathcal{X}, \mathcal{D}, A, \mathcal{F}, \bigotimes)$ *where*

- $\mathcal{X} = \{x_1, \ldots, x_4\}$
- $\mathcal{D} = \{D_i = \{true, false\}\}_{i=1}^{4}$
- $A = \mathbb{N}^{\infty}$
- $\mathcal{F} = \{h_1, h_2, h_3, p_1, p_2, p_3, p_4, p_{23}\}$
- $\bigotimes = +$

The global view of this cost network is $F(\mathcal{X}) = \sum_{f \in \mathcal{F}} f$. *We can express* $F(\mathcal{X})$ *extensionally with the following cost table,*

x_1	x_2	x_3	x_4	
f	f	f	f	∞
f	f	f	t	∞
f	f	t	f	10
f	f	t	t	6
f	t	f	f	∞
f	t	f	t	∞
f	t	t	f	5
f	t	t	t	∞
t	f	f	f	∞
t	f	f	t	8
t	f	t	f	∞
t	f	t	t	∞
t	t	f	f	∞
t	t	f	t	∞
t	t	t	f	∞
t	t	t	t	∞

Consider the WCSP problem $\mathcal{P} = (\mathcal{X}, \mathcal{D}, A, \mathcal{C}, \bigotimes, \bigoplus)$ *defined over cost network* $\mathcal{M}$*, where the marginalization operator* $\bigoplus$ *is* min*. Recall that since the marginalization operator is* min*,* P *is an optimization task and* $F(\mathcal{X})$ *is its objective function. The result of the WCSP problem* P *(i.e., the optimal cost of* $\mathcal{M}$*) is* $\min_{\mathcal{X}}\{F(\mathcal{X})\} = 5$*, which is the minimum among all the entries of* $F(\mathcal{X})$*. The corresponding optimal complete assignment is* $(x_1 = f, x_2 = t, x_3 = t, x_4 = f)$*.*

2.3.2 Belief Networks and Extensions

Belief networks [111], also known as bayesian networks, provide a formalism for reasoning about partial beliefs under conditions of uncertainty. They are used in a variety of applications including medical diagnosis, troubleshooting in computer systems, circuit diagnosis, traffic control, and signal processing.

The fundamental characteristic of belief networks is that functions are real valued in the interval $[0, 1] \subseteq \mathbb{R}$. The purpose of functions in belief networks is to specify conditional probabilities. Each function f_i contains a distinguished variable in its scope noted x_i. Function f_i denotes the probability of x_i conditioned to the remaining variables in $var(f_i)$.

Definition 2.3.9 *A* belief network *is a graphical model* $(\mathcal{X}, \mathcal{D}, [0, 1], \mathcal{P}, \times)$*, where* $\mathcal{X}$ *and* $\mathcal{D}$ *is a set of variables and its associated domains, respectively.* $\mathcal{P} = \{P_1, \ldots, P_n\}$ *is a set of real valued functions (i.e.,* $P_i : l(var(P_i)) \rightarrow [0, 1]$*), where* $P_i = P(x_i|var(P_i) - x_i)$ *are conditional probability tables (CPT) associated with each* x_i*. The combination operator is the multiplication* $\times$*.*

A belief network represents in a space efficient way a probability distribution over each complete assignment X, $P(X) = \prod_{i=1}^{n} P_i(X[var(P_i)])$.

There are two important reasoning tasks posed to belief networks: *belief updating* and *most probable explanation*. Both are based on some observations (i.e., instantiations) of some variables called *evidence*. Therefore, given a belief network $\mathcal{M}$ and an evidence e, both tasks are posed on $\mathcal{M}(e)$ (i.e., belief network $\mathcal{M}$ *conditioned* to the evidence e).

The first task, called belief updating, consists on computing the posterior marginal probability of *query* node(s) given some evidence.

Definition 2.3.10 *The* belief updating *problem of variable* x_i *when assigned to domain value* $a \in D_i$ *is a reasoning task* $\mathcal{B}(x_i, a) = (\mathcal{X}, \mathcal{D}, [0, 1], \mathcal{P}(e \cdot x_i = a), \times, +)$ *where* $\mathcal{P}(e \cdot x_i = a)$ *is the original set of CPTs* $\mathcal{P}$ *conditioned on the evidence* e *and on variable* x_i *taking domain value* a*. The marginalization operator is the* sum *over the reals.*

The belief updating of variable x_i is the result of computing

$$\underset{a \in D_i}{\forall} \mathcal{B}(x_i, a)$$

Similarly, the belief updating of a set of variables $\mathcal{Y}$ is the result of computing

$$\underset{\substack{x_i \in \mathcal{Y} \\ a \in D_i}}{\forall} \mathcal{B}(x_i, a)$$

The second main task, called most probable explanation, consists on finding a complete assignment which agrees with the evidence, and which has the highest probability among all such assignments.

Definition 2.3.11 *The* most probable explanation *(MPE) problem is a reasoning task* $(\mathcal{X}, \mathcal{D}, [0,1], \mathcal{P}(e), \times, \max)$ *where* $\mathcal{P}(e)$ *is the original set of CPTs* $\mathcal{P}$ *conditioned on evidence* e*, and the marginalization operator is* maximum*.*

Influence Diagrams [65] (also called *decision networks*) is a generalization of bayesian networks where not only probabilistic information on the environment, but also utilities expressing preferences and feasibility constraints on the decisions are captured. An influence diagram can be used to visualize the probabilistic dependencies in a decision analysis and to specify the states of information for which independences can be assumed to exist. The main task posed in an influence diagram is to find an assignment to the decision nodes that maximizes the expected utility.

A large number of frameworks have been proposed to model and solve such problems. In particular, the formalization of influence diagrams as a particular instantiation of the graphical model framework, aiming to capture locality of information and independence, was shown in [113, 112].

2.4 Graph Concepts and Complexity Implications

Essential to a graphical model is its underlying graph, called *interaction graph*. It captures the interdependency of the knowledge encoded in the graphical model. Roughly, the graph connects pairs of variables that interact directly in the problem.

Definition 2.4.1 *The* interaction graph *G of a graphical model* $(\mathcal{X}, \mathcal{D}, A, \mathcal{F}, \otimes)$ *is an undirected graph* $G = (V, E)$ *having a node for each variable (i.e.,* $V = \{i \mid x_i \in \mathcal{X}\}$*), and an edge between every two variables included in the scope of the same function (i.e.,* $E = \{(i,j) \mid \exists\, f \in \mathcal{F}$ s.t. $\{x_i, x_j\} \subseteq var(f)\}$*).*

Typical reasoning tasks over a graphical model $\mathcal{M}$ with n variables can be solved with depth-first search in space polynomial and time exponential on n (see Chapter 5). These trivial bounds may be improved by looking at the interaction graph of $\mathcal{M}$. This inspection allows us to gain insights into the structural properties of the model. In particular, there are two important features worth to check and exploit in the graph: its *degree of ciclicity* and its *connectivity*.

Consider connected graphs of a certain number of vertices. Clearly, the *trees* have the lowest level of ciclicity (they are acyclic), and *cliques* have the highest level of ciclicity (they have all the possible cycles). There are several measures of ciclycity that take into account the intermediate cases [57]. We consider the *induced width*.

Let $G = (V, E)$ be an undirected graph and o be an ordering of V, then:

Definition 2.4.2 *The width of node x_i subject to o, noted $w(o, x_i)$ is the number of adjacent to x_i which are before x_i in o. The* width *of G subject to o, noted $w(o)$, is the maximum width among the nodes.*

Definition 2.4.3 *The* induced graph *$G^*(o)$ is computed as follows: nodes are processed in reverse order, according to o. When processing node i, edges are added as needed in order to make a clique with all its adjacent which are still unprocessed.*

Definition 2.4.4 *The width of the induced graph is called the* induced width *and is noted $w^*(o)$. The minimum induced width over all its possible orderings is the induced width of a graph and is noted w^*.*

Finding the minimum induced width is NP-complete [35]. Observe that an obvious upper bound of w^* is $|V|$.

Example 2.4.1 *Figure 2.2 (a) depicts the interaction graph G of the cost network in example 2.3.3. Figure 2.2 (b) shows the induced graph $G^*(o)$ along ordering $o = (x_1, x_2, x_3, x_4)$. $G^*(o)$ has one new edges. The new edge,*

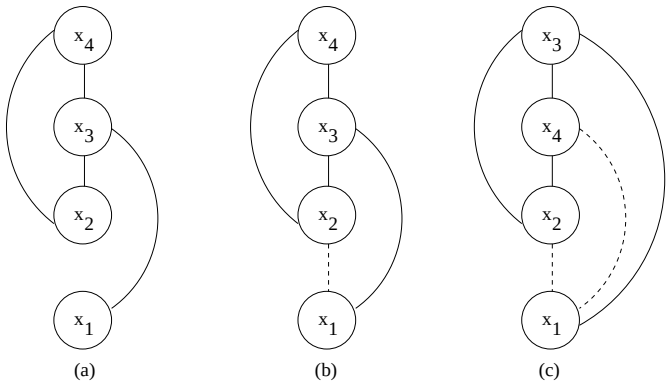

Figure 2.2: Interaction graph and induced graphs.

between x_2 and x_1 (doted line), is added when processing x_3. The induced width $w^*(o) = 2$. Figure 2.2 (c) shows the induced graph $G^*(o)$ along ordering $o = (x_1, x_2, x_4, x_3)$. $G^*(o)$ has two new edges (doted lines), both added when processing x_3. The induced width $w^*(o) = 3$.

Observation 2.4.1 *Most reasoning tasks on graphical models can be solved in time and space exponential on the induced width of the interaction graph using* inference *(see Chapter 7).*

A related structural parameter is the bandwidth. Let $G = (V, E)$ be an undirected graph and o be an ordering of V. The bandwidth of an ordering is the maximum distance, according to this ordering, between two connected vertices.

Definition 2.4.5 *The* bandwidth *of a graph G [149] is the minimum bandwidth on all its possible vertex orderings.*

For our purposes, bandwidth is important because it is always greater than the induced width.

The second main property is the graph connectivity. In an undirected graph G, two vertices u and v are called connected if G contains a path from

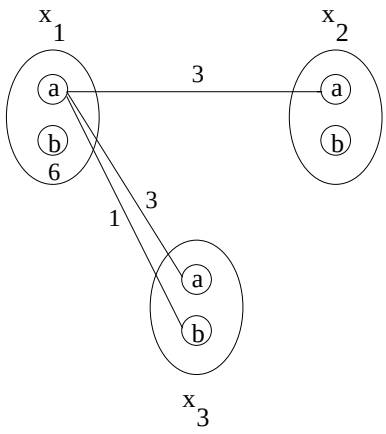

Figure 2.3: Micro-structure of a graphical model.

u to v. Otherwise, they are called disconnected. A graph is called *connected* if every pair of distinct vertices in the graph is connected. A *connected component* is a maximal connected subgraph of G.

Observation 2.4.2 *Most reasoning tasks on graphical models can be solved by solving independently each connected component. The global solution is trivially computed using the combination operator* $\bigotimes$ *of the model. Note that, in this case, the complexity of solving the entire reasoning task is given by the worst complexity among the different connected component.*

Example 2.4.2 *Consider a graphical model* $\mathcal{M}$ *with underlying interaction graph* $G = (V, E)$ *such that* $G = G_1 \cup \ldots \cup G_k$*, where each* $G_i = (V_i, E_i)$ *is a connected component. Typical reasoning tasks over* $\mathcal{M}$ *can be solved in time exponential over* $\max\{|V_1|, \ldots, |V_k|\}$ *and space polynomial on* $|V|$*. The time complexity can be improved as a trade-off increasing the space complexity. Let* w_i^* *be the induced width of* G_i*. Then, typical reasoning tasks over* $\mathcal{M}$ *can be solved in time and space exponential on* $\max\{w_1^*, \ldots, w_k^*\}$.

Very often, it is useful to look at the interdependencies between the domain values of variables. To that end, the interaction graph can be further

extended with its *micro-structure*. The micro-structure of the interaction graph is as follows. As before, each variable is represented by a node. The domain values of each variable are depicted circled inside the corresponding node. Only valuations different from $\bot$ are indicated. Unary valuations coming from unary functions are indicated below each domain value. N-arity valuations coming from n-arity functions are indicated as weighted edges connecting the n dependant domain values. Consider the micro-structure of Figure 2.3. It represents a cost network with three variables $\{x_1, x_2, x_3\}$ and two domain values per domain $D_i = \{a, b\}$. The set of functions is composed by two binary functions $f_{x_1x_2}(x_1, x_2)$ and $f_{x_1x_3}(x_1, x_3)$ such that

$$f_{x_1x_2}(a, a) = 3, f_{x_1x_2}(a, b) = f_{x_1x_2}(b, a) = f_{x_1x_2}(b, b) = 0$$

$$f_{x_1x_3}(a, a) = 3, f_{x_1x_3}(a, b) = 1, f_{x_1x_3}(b, a) = f_{x_1x_3}(b, b) = 0$$

and one unary function $f_{x_1}(x_1)$ such that $f_{x_1}(a) = 0$ and $f_{x_1}(b) = 6$.

Chapter 3

Related Work

The purpose of this chapter is to overview the related state-of-the-art. It is divided into two sections. The first section is devoted to mono-objective optimization techniques in graphical models. So it includes the algorithms that are the source of inspiration for our work. It covers search, inference, and lower bounding algorithms. The second section is devoted to multi-objective optimization algorithms. So it includes work related to ours. It covers two different areas: Artificial Intelligence and Mathematical Programming. We do not attempt to be exhaustive, but only to give a comprehensive introduction to the most relevant topics.

3.1 Optimization in Graphical Models

Efficient algorithms for mono-objective optimization tasks in graphical models (e.g., WCSP or MPE problems) have been exhaustively studied during three decades. They can be roughly divided into two general types: search and inference.

Search algorithms (e.g., depth-first branch-and-bound, best-first branch-and-bound) solve the problem by a sequence of variable assignments. In its traditional form, they search a tree such that each tree level corresponds to a problem variable and different tree nodes correspond to the different

assignment alternatives. The main disadvantage of this tree structure is that it does not exploit the independences represented in the underlying graphical model. More sophisticated search tree schemas (e.g., AND/OR trees) have been proposed in order to address this issue.

Inference algorithms (e.g., Variable Elimination, Tree Clustering) solve the problem by a sequence of transformations. Each transformation generates an equivalent but simpler problem. Roughly, the idea is that the new problem contains explicitly some knowledge that was only implicit in the initial problem.

In the following we review these two approaches.

3.1.1 Search

Search in graphical models is particularly adequate because the set of all total assignments can be naturally expressed as a tree with bounded depth. Therefore, there is no danger to get lost in infinite branches. The set of all total assignments is the *search space*.

Traditionally, the search tree is generated with the following procedure based on a predefined or static ordering among variables and domain values. The root of the tree is the empty assignment. Its children are the m possible domain values of the first variable. For each node, its children are the m possible domain values of the second variable, and so on. If the process is done for the n variables, it generates a tree with depth n such that each tree level corresponds to a variable and each node in that level corresponds to its assignment to one of its domain values. Figure 3.1 (b) shows this tree for the problem of example 2.3.3 (see interaction graph in Figure 3.1 (a)) along ordering $o = (x_3, x_4, x_2, x_1)$. Note that, in this case, the order in which variables and domain values are selected were established before search and is respected throughout all the search. However, it can be set dynamically during search without losing its completeness. Thus, using dynamic variable and domain value orderings the next current variable and the order in which its domain values are assigned is decided at each search node. Figure 3.1 (c)

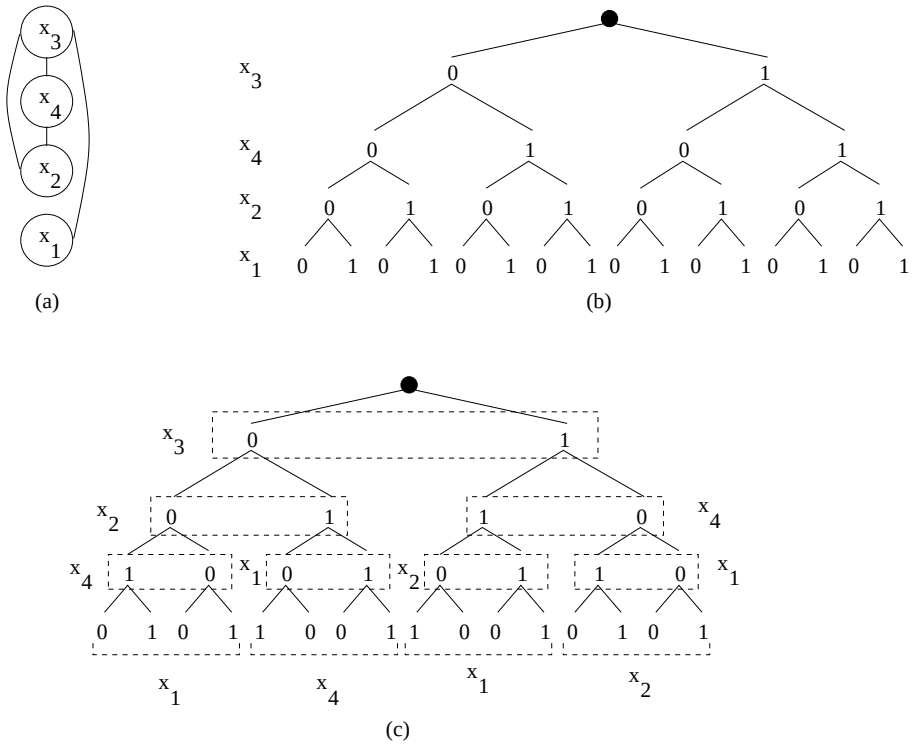

Figure 3.1: Constraint graph of a graphical model with 4 variables and 2 domain values per variable and two possible search trees.

shows the search tree for the same example along a possible dynamic variable and domain value orderings. Note that, in this case, search trees are more flexible. Since the children of a certain node represent the set of alternatives for the chosen variable, we will refer to this type of trees as *OR trees*.

Each internal node in an OR tree corresponds to a partial assignment. Following any path from the tree root to a leaf, each step extends the partial assignment (initially empty) including one more variable. Thus, the internal nodes are subproblems where only the remaining variables have to be assigned and a solution is a path from the tree root to a leaf. As a consequence, solving an optimization problem consists on finding a path with the minimum associated cost.

Branch-and-bound is the main search schema to solve mono-objective optimization problems. It traverses the search tree while keeping the cost of the best solution found so far, which is an upper bound of the optimal solution. The upper bound allows the search to use a bound strategy in order to prune unfeasible branches. In each node branch-and-bound computes an underestimation or lower bound of the best solution that can extend the current node. When this lower bound is greater or equal than the current best solution, the algorithm has discovered a branch that cannot deal to a better solution and can be discarded. As a consequence, the current search branch is pruned. The order in which branch-and-bound visits the nodes determines the particular search algorithms (e.g., *depth-first branch-and-bound search* (DF-BB) [53] and *best-first branch-and-bound search* (BF-BB)). In DF-BB, the search begins at the root node and explores as far as possible along each branch. Namely, DF-BB explores one path at each step. In BF-BB, the search begins at the root node and *expands* (i.e., generates the children of) the most promising node chosen according to some heuristic function. Namely, BF-BB may explore a different path at each step. The time complexity of both search schemas is exponential in the number of variables of the problem. However, the space complexity of DF-BB is linear in the number of variables, while BF-BB is also exponential in the number of variables for graphical models. DF-BB is the usual algorithm of choice (see Chapter 5 for more details).

OR trees do not exploit the independences among variables. Consider the example in Figure 3.1 (a). As clearly shown in the interaction graph, variable x_1 only interacts with variable x_3. Once x_3 is assigned, subproblems composed by x_1, and x_4 and x_2, respectively do not interact. Therefore, given each assignment of x_3, the search space it roots is decomposed into two independent subproblems. Each independent subproblem could be then solved independently. However, its associated OR search tree in Figure 3.1 (b) does not take into account such independences. Instead, it considers the assignment of x_1 for each different assignment of x_4 and x_2. Since incompat-

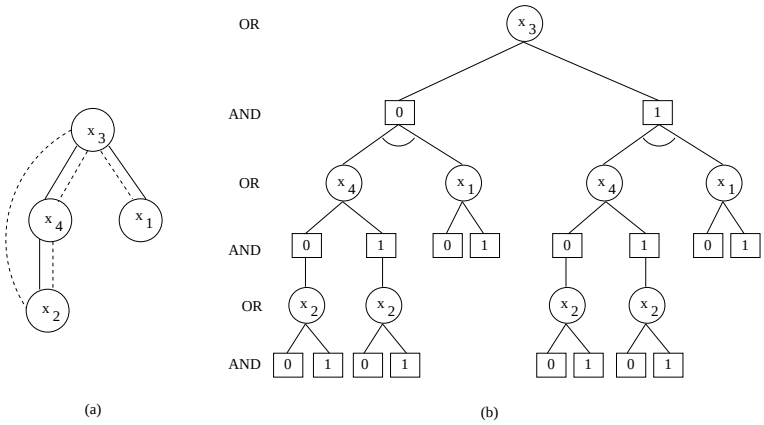

Figure 3.2: AND/OR search tree.

ibilities in the assignment of x_1 comes from the assignment of x_3, the search algorithm wastes time until changing the assignment of x_3. An alternative to address this issue is to generate the search space following a tree-like ordering of the variables, called pseudo-tree.

A *pseudo tree* arrangement of an interaction graph G [52] is a rooted tree with the same set of vertices as G and the property that adjacent vertices in G must be in the same branch of the rooted tree. Figure 3.2 (a) shows one of the many pseudo-tree arrangements for our running example. Solid lines indicate the edges in the tree. Dotted lines are the edges in the original interaction graph. The pseudo tree highlights the independences among variables. Note that there is no edge (i.e., dependency) between branches. Therefore, if a given node has $q > 1$ children in the pseudo-tree, once the variables in its path to the root node are assigned, the subproblem it roots is divided into q independent subproblems. These subproblems can be solved independently.

The search tree associated with a given pseudo tree is called AND/OR search tree. An *AND/OR search tree* is a tree structure with alternating levels of AND and OR nodes. The *OR nodes* are labeled x_i and correspond to variables. The *AND nodes* are labeled $\langle x_i = a_j \rangle$ and correspond to the assignment of variable x_i with value $a_j \in D_i$. The procedure to generate an

AND/OR search space is as follows. The root of the AND/OR search tree is an OR node labeled by the root node of its associated pseudo tree. The children of the OR nodes are AND nodes labeled with assignments $\langle x_i = a_j \rangle$ for all $a_j \in D_i$. The children of an AND node $\langle x_i = a_j \rangle$ are OR nodes labeled with the children of variable x_i in the pseudo tree.

Example 3.1.1 *Consider our running example. Figure 3.2 (b) shows the AND/OR search space under the pseudo tree shown in figure 3.2 (a). The equivalent traditional search space is given in figure 3.1 (b), where AND levels are omitted. Note that the underlying pseudo-tree in the traditional search tree is a chain organization of its variables.*

In AND/OR search spaces, OR nodes represent alternative ways of solving the problem while the AND nodes represent problem decomposition into independent subproblems, all of which need to be solved. Therefore, a solution is not a path but a subtree of the AND/OR search tree. The *solution subtree* contains the root node of the AND/OR tree and, for every OR node it contains one of its children and for each of its AND node it contains all its children. As a consequence, solving an optimization problem consists on finding a solution subtree with the minimum associated cost.

Depth-first AND/OR search [37] traverse depth-first AND/OR trees in order to find the best solution subtree. Since each node roots an independent subproblem, solving a problem is solving the independent subproblems it roots and *using* their valuations in a convenient way. The manner they are used depends on the kind of node. OR rooted problems *join* the solutions found in its subproblems using the $\oplus$ operator over valuations. AND rooted problems *aggregate* the solution obtained for each of its child subproblems using the $\otimes$ operator over valuations. The algorithm solves each subproblem progressively, from leaf nodes to the root. Each node is labeled with the optimal cost of the subproblem it roots until the root node is labeled. Then, the whole problem is solved and its optimal cost is the one hold in the root node. The efficiency of the algorithm can be also improved thanks to a

bounding evaluation function which underestimates the best-cost solution of the remaining search tree. The time complexity is $O(exp(m))$, where m is the depth of the AND/OR search tree. Note that the depth of the AND/OR search tree is the same as the depth of its underlying pseudo-tree. The minimal depth m over all pseudo trees satisfies $m \leq w^* \log n$, where w^* is the induced width of the interaction graph of the graphical model, and n is the number of variables [8, 20]. Note that the time complexity over a traditional search tree along any ordering is $O(exp(n))$. Therefore, an AND/OR search tree may sometimes reduce the search time exponentially. The space complexity is linear in the number of variables.

Recursive conditioning (RC) [30] and *Value Elimination* [7, 6] are similar algorithms described for bayesian networks. Although the algorithms use different notation, its essence is the same as for depth-first AND/OR search, as demonstrated in [37].

AND/OR search algorithms can trade space for time using *caching* schemas. As we have said, each search node roots a subproblem. Different nodes may root identical subproblems. The idea of caching is to store the solutions of already solved identical subproblems. Therefore, each subproblem will be solved only once. At each search node, the algorithm checks whether that subproblem was already solved. If that is the case, it retrieves the stored solution and does not solve it again. The more solutions the algorithm stores, the less subproblems will be solved more than once and the more space will be needed to store them. Therefore, there is a trade-off between the space and the time of the algorithm. We refer the interested reader to [101, 30] for more details.

3.1.2 Inference

Inference algorithms (also known as *decomposition* methods) solve a problem by a sequence of transformations that create simpler equivalent problems. These transformations make explicit some knowledge that is implicit in the network. Two problems are equivalent if they have the same set of solu-

tions. The inference process generates equivalent problems with respect to the original one, but easier to solve.

Usually, complete inference algorithms simplify problems by e.g. eliminating variables. These methods process (eliminates) variables in a certain order and infer new dependencies among the remaining variables of the problem. We focus on *bucket elimination*, which is a unifying algorithmic framework that generalizes variable elimination algorithms to accommodate many complex problem solving and reasoning tasks, as mono-objective optimization.

Bucket elimination (BE) [15, 34] process variables one by one in a given order. Processing a variable means generating an equivalent representation that excludes, or eliminates that variable. The elimination of variable x_i is done as follows. First, the algorithm generates its associated *bucket*, which contains all the functions defined on variable x_i. Next, BE computes a new function by combining (via $\bigotimes$) all the functions in that bucket and eliminating x_i (via $\bigoplus$). The new function summarizes the effect of x_i on the rest of the problem. Therefore, the graphical model can be updated by removing the functions in the processed bucket and adding the new function. The problem is simplified because the eliminated variable disappears from the problem. Moreover, it is equivalent as it preserves the optimal cost. The algorithm terminates when all variables are processed. The elimination of the last variable produces an empty scope function (i.e., a constant) which is the optimal cost of the problem. A more detailed description will be given in Chapter 7.

It can be shown that the complexity of bucket elimination algorithms is time and space exponential in $w^*(o)$. Clearly, the induced width will vary with the variable ordering. Although finding a minimum induced width ordering is NP-hard, good heuristic algorithms are available [35].

It has been recently discovered that AND/OR search with full caching is essentially equivalent to BE [37]. There is a solid line of current research that investigates the potential benefits of this new perspective of BE (e.g. dynamic variable and value orderings, data structures, etc).

Another line of fruitful research is the specialization of inference algorithms to particular frameworks. For instance, the idempotency of constraint networks (i.e. implicit constraints can be safely added to the network) has allowed the improvement of complexity bounds for the CSP problem (e.g. Hyper-tree decompositions [57]). Recently, the instantiation of BE to Max-SAT has been re-discovered as a natural extension of directional resolution [31, 21, 90]

3.1.3 Lower Bounds

As we saw in Section 3.1.1, search algorithms for optimization problems follow the branch-and-bound strategy. These algorithms need to compute a lower bound at every visited search node. It is well known that the quality of the lower bound is central to the pruning power of the search algorithm. However, better lower bounds are usually more expensive to compute and the overhead may not pay-off the pruning gains. The goal then is to find parameterized families of lower bounds where the parameter is used to control the accuracy and the computation cost. There are two prominent types of lower bounding algorithms: local consistency and mini-bucket elimination.

Local consistency is a family of increasingly harder properties about the problem [104, 96, 51, 89, 32, 27]. The control parameter is the size of the subnetwork involved. The more variables involved, the harder the property is. The simplest form of local consistency is *node consistency*, which only takes into account unary functions. The next local consistency is *arc consistency*, which takes into account binary functions. In general, *i-consistency* takes into account functions with $i + 1$ variables in its scope. Each local consistency property comes with its enforcing algorithm, which works in polynomial time.

In weighted constraint networks, the effect of the transformation is a *movement* of costs. For instance, the enforcement of arc-consistency produces a *movement* of costs from binary to unary and zero-arity functions. The zero-arity function turns to be a lower bound of the optimal cost. The idea is to enforce local consistency at each node during search. If the current node

cannot lead to a better solution and this situation is not detected by the search algorithm, achieving some level of local consistency may lead to its discovery. In this way, search does not need to unsuccessfully visit deeper nodes of the current subtree. All changes made by the local consistency enforcing algorithm in the current subproblem remain in its children, so the local consistency does not have to be computed from scratch at every node. As far as local consistency enforcing preserve the problem semantics, this schema is valid with any search strategy, from the usual depth or best first, to more sophisticated ones.

Another lower bounding algorithm is mini-bucket elimination. *Mini-bucket elimination* (MBE) [38] is the approximation version of BE. By eliminating a variable, BE makes explicit the impact of this variable on the rest of the variables. Instead, MBE eliminates a variable from restricted subsets of these constraints, thus reducing the computation. The result is an approximation of the optimal cost. MBE has a control parameter z which indicates the level of accuracy. The higher the value of z, the tighter the approximation obtained. In the limit, when z equals the number of variables n, MBE is equivalent to BE and, as a result, it computes the optimal cost. The time and space complexity is exponential in the parameter z. Therefore, there is a trade-off between the accuracy and resources. Further details will be given in Chapter 8.

3.2 Multi-objective Optimization

Multi-objective optimization has been studied in different research areas and from different points of view. In this Section, we review some of the techniques used to solve this kind of problems in *Artificial intelligence* and *Operations research*.

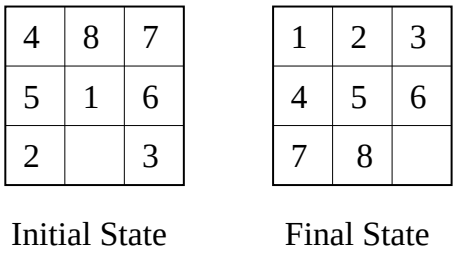

Figure 3.3: A possible initial state and the final state of the 8-puzzle problem.

3.2.1 Artificial Intelligence

Although mono-objective optimization has been studied in depth within AI, there is little work on multi-objective optimization. In the following we review the main contributions in *heuristic search* and *constraint programming*. Each one of this research fields considers *slightly* different types of problems.

Heuristic Search

Heuristic search has traditionally been one of the fundamental problem solving tools in AI. It considers problems modeled as a set of configurations or states that the problem can be in. The set of states, called *state space*, in general form a graph where two states are connected if there is an operation that can be performed to transform the first state into the second. Typically, each taken action has an associated *cost*. The *path cost function* combines in a certain way the costs of the actions in a path. The task is to compute the *best* path from an initial state to a goal state with a desired property according to a given path cost function.

Example 3.2.1 *The 8-puzzle problem is a well-known mono-objective example. It consists of a 3×3 board with eight numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The objective is to reach a specified goal state from a start state, as the ones shown in Figure 3.3. The standard formulation is as follows:*

- **States**: *a state description specifies the location of each of the eight tiles and the blank in one of the nine squares.*
- **Initial state**: *any state can be designated as the initial state.*
- **Actions**: *the possible set of actions is the move of blank to left, right, up, or down.*
- **Goal state**: *any state can be designated as a goal state.*
- **Path cost**: *each step costs 1, so the path cost is the number of steps in the path.*

Multi-objective optimization problems are modeled in the same way, the only different being that the cost of an action is a vector (each component corresponds to an objective). As a result, the cost of a path is also a vector. Since vectors are partially ordered, there does not exist an unique *best* path, but a set of *incomparable* or *non-dominated* paths. Then, the task is to compute the set of *non-dominated best* paths, according to a given path function.

Example 3.2.2 *Consider a road map where each road between two cities has two associated values: the* distance *between the cities and the* driving-time*. The task is to find the shortest and quickest way to get from one location to another. We can formulate this problem as,*

- **States**: *the cities.*
- **Initial state**: *a given city.*
- **Actions**: *driving from one city to another, if there is a road that links them.*
- **Goal state**: *a given city.*

- **Path cost**: *each step costs* (d_{ij}, t_{ij})*, where* d_{ij} *and* t_{ij} *is the distance and driving-time to from city* i *to city* j*, respectively. The cost of a path is the pairwise sum of each step cost.*

These problems are solved by searching through the state space. The most important mono-objective algorithms are A^*, *IDA**, and *frontier search*. These algorithms have been extended to multi-objective optimization. In the following, we outline these extensions.

Multiobjective A*. A^* [60, 61, 106, 110] is a best-first search algorithm that finds the least-cost path from a given initial state to one goal state (out of one or more possible goals). The basic operations of the algorithm are node selection and expansion at each iteration. Nodes are selected according to a given evaluation heuristic function (usually denoted $f(x)$). In A* each open node is associated to a single path that can be further expanded. The main limitation of A* is its exponential space requirements.

Multiobjective A^* (MOA*) [139] maintains the structure and basic operations of A* accommodated to the new multiobjective context. Each open node is now associated with a set of non-dominated paths. The heuristic function, noted $f(x)$, is a vector of heuristic functions. Nodes are selected according to $f(x)$ and another domain specific heuristic function to break ties. When a node is selected for expansion, all known non-dominated paths reaching that node are extended.

Mandow and Pérez [97] propose NAMOA*, a new approach to MOA*. The algorithm considers path selection and expansion as basic operations. It maintains a list of open paths, instead of open nodes. Then, individual paths are selected for expansion. This strategy results in a more efficient space management since only the paths that can lead to new non-domination solutions are expanded.

Multiobjective Iterative Deepening A*. *Iterative deepening* A^* (IDA*) [83] is a space-efficient refinement of A*. The idea of the algorithm is to repeatedly increase a threshold value and to perform depth-first search where the objective function is bounded by the threshold. The threshold is in-

creased until the algorithm finds a solution. The main advantage of IDA* over A* is its polynomial space complexity. The main disadvantage is the repeated expansion of nodes. In a tree, or a graph with very few cycles, a IDA* search is usually the best choice.

*Multi-objective iterative deepening A** (IDMOA*) [59] is the generalization of IDA* to the multi-objective context. In IDMOA* the threshold is a vector where each component is the threshold value for each objective function. First, the algorithm finds the best solution according to the first objective function. Then, the algorithm sequentially adjust the threshold of each objective function and finds the non-dominated solutions that do not surpass this threshold. The set of solutions found in each search is used to set the maximum threshold for each objective. The algorithm terminates when all the objectives have been considered. A similar strategy has been also proposed in the operations research field under the name of ϵ-constraint (see Section 3.2.2, Figure 3.4).

Constraint programming

Constraint programming (CP) [3] is a research field whose main goal is the development of languages and algorithms to model and solve problems that can be expressed as a constraint satisfaction problem (or any of its extensions) (see Section 2.3.1). One approach to solve mono-objective optimization problems with constraint programming technology is to replace the objective function $F(\mathcal{X})$ by a constraint $F(\mathcal{X}) < K$ where K is initially set to a sufficiently large number. Each time a solution is found, the constraint is tightened by decreasing the value of K. The process goes one until the solver reports failure. The last value of K is the optimum. This algorithm is a variation of IDA* in which the threshold decreases instead of increasing. This can be done because the search tree has bounded depth..

The efficiency of a CP solver depends on its ability to model and propagate constraints. When solving mono-objective optimization problems, that means its ability to express and propagate $F(\mathcal{X}) < K$. We can say that

current CP solvers are only efficient for very specific forms of $F(\mathcal{X})$ (e.g., linear functions).

Gavanelli [54] extends this idea to multi-objective optimization. His algorithm maintains the set of non-dominated solutions found so far because they are candidates to be in the problem's efficient frontier. Moreover, every time the algorithm finds a new solution, it explicitly adds a set of constraints that limit the next solutions to be better, in the non-dominated sense, than the already achieved ones. In other words, a (tentative) possible solution will be pruned off if it is worst in all the objective functions than an already obtained solution. In some sense, this constraints *mimics* the role of the upper bound and the pruning condition in the mono-objective case. However, since the algorithm solves a multi-objective problem as a sequence of constraint satisfaction problems, these two concepts are hidden. Moreover, the algorithm does not detail the lower bounding used.

Junker [72] studies the potential of constraint programming in preference-based optimization problems. His main concern is the preference elicitation process of a user that may have multiple criteria in mind. His proposal consists in solving a sequence of mono-objective optimization problems with which the user refines his preferences. The procedure, called *preference-based search* (PBS), consists of two modules: a master-PBS explores the criteria in different orders and assigns optimal valuations to them. The optimal valuation of a selected criterion is determined by a sub-PBS, which performs a mono-objective branch-and-bound search through the original problem space. Through successive mono-objective optimizations, the algorithm explores the whole search space and finds the non-dominated solutions.

Other works related to Junker's study the transformation of the partial order defined by multi-objective optimization problems to a total order. That is the case of *leximim preorder* [105] used in *social choice theory* [76] or the order induced by the *Choquet integral aggregation function* used in *multi-attribute utility theory* (MAUT) [76]. In the first case, Bouveret and Lemaitre [22] introduce three different generic algorithms based on branch-and-bound

that finds the optimal solution in terms of the leximin preorder. In the latter, Le Huédé et. al [66] introduce *MCS*, a general search branch-and-bound algorithm that alternates mono-objective searches following various mono-criterion strategies to find solutions of increasing quality with respect to the aggregation function.

3.2.2 Operations Research

In the Operations Research (OR) field, problems are mathematically modeled as a set of real or integer variables and a real objective function subject to some constraints. The set of feasible assignments, noted $Q \subset \mathbb{R}^n$, is called *feasible set*. Usually, Q is given implicitly through constraints in the form of mathematical functions, i.e., $Q = \{X \in \mathbb{R}^n | g_j(X) \leq 0, j = 1, \ldots, l;\ h_j(X) = 0, j = 1, \ldots, m\}$, where X is a complete assignment. In the case of multi-objective optimization, the objective function is a real vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^p$ composed of p real-valued objective functions, $f = (f_1, \ldots, f_p)$, where $f_k : \mathbb{R}^n \to \mathbb{R}$ for $k = 1, \ldots, p$. Formally, the problem is stated as

$$\min(f_1(X), \ldots, f_p(X)) \texttt{ subject to } X \in Q$$

The symbol min is generally understood as finding optimal values of f in terms of the usual non-domination partial order. In this context, the set of optimal vectors is called *solution set*.

There are two general approaches to generate the solution set of multi-objective optimization problems: *scalarization* and *non-scalarization methods*. These approaches convert the multi-objective problem into one mono-objective problem, a sequence of mono-objective problems, or another multi-objective problem. Under some assumptions solution sets of these new programs yield solutions of the original problem. Scalarization methods explicitly employ a scalarizing function that combine in a certain way the multiple objectives into one single objective. Non-scalarizing methods transform the original problem using other methods. Solving the mono-objective problem

typically yields one solution of the original problem so that a repetitive solution scheme is needed to obtain a subset of solutions of the multi-objective problem.

The *weighted sum* approach [56, 67, 41] is one of the most popular methods based on scalarization. It is based on the aggregation of the different objectives via a weighted sum. The problem solved is,

$$\min\{\sum_{j=1}^{p} \lambda_j f_j(X) : \ X \in Q\}$$

where $0 \leq \lambda_j \leq 1$ and $\sum_{j=1}^{p} \lambda_j = 1$. Different weighted vectors $(\lambda_1, \ldots, \lambda_p)$ would ideally lead to finding different elements of the solution set, but for an unknown problem it is not clear what weight combination to choose. Even if all possible weight combinations were used, it cannot be guaranteed to find Pareto-optimal solutions in concave regions of the solution set.

The ϵ-*constraint* approach [24] is another traditional scalarization method to generate the whole solution set. In this method one objective function is retained as a sclalar-valued objective while all the other objective functions are bounded from below by means of additional constraints. An intermediate ϵ-constraint sub-problem is formulated as

$$\begin{array}{rl} \text{minimize} & f_k(X) \\ \text{subject to} & f_i(X) \leq \epsilon_i, \ i = 1, \ldots, p, \ i \neq k \\ & X \in Q \end{array}$$

By a systematic modification of the constraint bounds ϵ_i, the algorithm obtains different elements of the solution set. The method relies on the availability of a procedure to solve single-objective optimization problems.

Since we use this method in some of our experiments on bi-objective problems, Figure 3.4 shows an implementation for the case of two objectives. The algorithm receives two objective functions f_1 and f_2, and their corresponding lower and upper bounds (l_1, l_2) and (u_1, u_2), respectively. The algorithm executes a sequence of minimizations of f_1 subject to different constraints

function ϵ-`constraint`$((f_1, f_2), (l_1, l_2), (u_1, u_2))$
return set of pairs
1. $\mathcal{E} := \emptyset$;
2. $\epsilon_1 := u_1; \epsilon_2 := l_2$;
3. **do**
4. $\quad X :=$ `minimize`(f_1) `subject to` $f_2 < \epsilon_2$ **and** $f_1 < \epsilon_1$;
5. $\quad$ **if** $\nexists (v_1, v_2) \in \mathcal{E}$ `such that` $(v_1, v_2) < (f_1(X), f_2(X))$ **then**
6. $\quad\quad \mathcal{E} := \mathcal{E} \cup \{(f_1(X), f_2(X))\}$;
7. $\quad$ **endif**
8. $\quad \epsilon_1 := f_1(X)$; $\epsilon_2 := \epsilon_2 + 1$;
9. **while** $\epsilon_1 > l_1$ **and** $\epsilon_2 < u_2$;
10. **return** $\mathcal{E}$;

endfunction

Figure 3.4: ϵ-constraint algorithm for two objective functions.

on $f_2 < \epsilon_2$ (line 4). The first constraint is $f_2 < l_2$ (line 2) and iteratively increasing ϵ_2 and decreasing ϵ_1 using the valuation of f_1 on the optimum of the previous single-objective run (line 8). The output of the algorithm is the Pareto set defined by the objective function $f = (f_1, f_2)$.

The lexicographic approach [39, 40, 10] is one of the most popular non-scalarazing methods. It assumes a ranking of the objective functions according to their importance. The lexicographical total order is defined as: $f(X) <_{lex} f(X')$ if $f_j(X) < f_j(X')$, where j is the smallest index such that $f_j(X) \neq f_j(X')$. The lexicographic problem can be solved with respect to one, or all permutations of the objective functions f_j.

There are some specially structured problems for which dedicated algorithms have been proposed. The most important case are the *multi-objective linear problems* (MOLP), where both objective functions and constraint functions are linear. Formally, a MOLP is

$$\begin{array}{rl} \min & Cx \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

where C is a $p \times n$ objective function matrix, A is an $l \times n$ constraint matrix, and $b \in \mathbb{R}^l$. It is usually assumed that the rows of A are linearly independent. This problems can be solved with *Multi-objective Simplex methods* [43, 42, 68, 138, 4], an extension of the Simplex methods used for solving mono-objective linear problems. They proceed in three phases. Roughly, first, an auxiliary mono-objective linear problem is solved to check feasibility. There exists a solution if and only if the optimal value of this linear problem is 0. If that is the case, phase 2 founds an initial extreme point or the algorithm stops with the conclusion that the solution set is empty. Finally, phase 3 explores all efficient extreme points or efficient bases. In order to determine the whole solution set, it is necessary to find subsets of efficient extreme points, the convex hulls of which determine maximal efficient faces.

When it is difficult or impossible to obtain the whole Pareto set due to the computational effort involved, one may use heuristic approaches for approximating the solution set. The most widely studied approximation methods are the so called *population-based* metaheuristics, which maintain a while set of solutions (the population) and try to evolve the population toward the solution set. Many different techniques, described in the literature as evolutionary and genetic algorithms (see [25, 48, 41] for a detailed review), have been developed to evaluate the fitness of individual solutions in a multi-objective context and guarantee enough diversity to achieve a uniform distribution of solutions over the whole solution set. Research on this topic was initiated by Schaffer's *vector evaluated genetic algorithm* (VEGA [128]). Other important references in this area include Fonseca and Fleming with the *multi-objective genetic algorithm* (MOGA [47]), Srinivas and Deb with the *nondominated sorting genetic algorithm* (NSGA [137]), Horn et al. with the *niched pareto genetic algorithm* (NPGA [64]), Knowles and Corne with the *pareto archived evolution strategy* (PAES [79]) and Zitler and Thiele

with the *strength pareto evolutionary algorithm* (SPEA [150]).

Chapter 4

Using Semirings to model Multiobjective optimization[1]

The main objective of this Thesis is to solve multi-objective optimization problems under the graphical model framework. A first necessary step that we address in this chapter is to rephrase multi-objective optimization with graphical models terminology. More precisely, if P_{mo} is a multiobjective problem we need to define a graphical model and an optimization task able to compute the efficient frontier of P_{mo}. We achieve that goal by combining two types of constructors: the *p-composition* and the *frontier algebra*. Namely, if we combine the p objectives of P_{mo} via p-composition and extend the result via its frontier algebra, the resulting optimization problem computes the efficient frontier of P_{mo}. Note that the frontier algebra is an original constructor never used before.

We develop our formalization inside of the so-called *semiring CSP* (SCSP) framework. In other words, we show that our formalization is built over an algebraic structure that satisfies all the axioms that the SCSP framework assumes as necessary for graphical models. For the sake of generality, we relax (only in this Chapter) our assumption of additive objective functions.

[1]The contributions of this Chapter have benefited from discussion with Stefano Bistarelli and Fabbio Gadduci.

Namely, we consider problems with more than one objective function where the only requirement is that the independent optimization of each objective function can be expressed as a SCSP problem. In this way, we depict our contributions in the most general context.

The structure of this Chapter is the following: In the next section we review the main algebraic frameworks that have been proposed in the literature to axiomatize graphical models. As we show, the semiring-based approach seems to be the best option for our purposes because it can deal with partially ordered optimization problems. Then, we introduce the two semiring constructors that we need to model multi-objective problems: p-composition and frontier algebra. Next, we show that given a multiobjective problem defined on p totally ordered c-semirings, the frontier algebra of their p-composition can be used to model the computation of its efficient frontier. Finally, we instantiate the framework to the case in which all the individual c-semirings are additive. Such instantiation, that we call multi-objective weighted CSPs (MO-WCSP), is especially important because it will be used in all the subsequent chapters.

4.1 Algebraic Frameworks for modeling Optimization Tasks

Consider a reasoning task $\mathcal{P} = (\mathcal{X}, \mathcal{D}, A, \mathcal{F}, \bigotimes, \bigoplus)$. Its valuation structure is the triple $\mathcal{K} = (A, \otimes, \oplus)$ (see Section 2.1). In the previous chapter, valuation structures were defined in a very *imprecise* form, that is, without justifying the properties that they must satisfy. However, the behaviour of valuation structures of the most usual tasks implicitly give us the desired common properties for each operator. In the context of optimization:

- The set A is used to specify how good or bad are the different assignments of the variables. This means, that there must be an order $\leq_{\mathcal{K}}$ to compare the values (i.e., $\forall\, a, b \in A$, $a \leq_{\mathcal{K}} b$ means that a is better than

b).

- The operator $\otimes$ is used to combine valuations from the different functions.
  - Since graphical models are defined by a set of functions, the order in which those functions are combined has to be irrelevant. This means that the $\otimes$ operator must be *associative* and *commutative*.
  - The combination of valuations must produce higher valuations. Formally, this means that the $\otimes$ operator must be *monotone* (i.e., $\forall\, a, b \in A, \quad a \leq_{\mathcal{K}} a \otimes b$). This property is heavily exploited by algorithms and plays a key role in their performance.
  - For modeling purposes, it is convenient that A contains a special value $\top$, called *top*, expressing the notion of *total dislike* or *inconsistency* of an assignment. Any tuple receiving such valuation from a function should immediately be identified as inconsistent. Consequently, $\top$ must be the *absorbing* element of $\otimes$ ($\forall\, a \in A, a \otimes \top = \top$).
  - Similarly, it is convenient to have a value $\bot$, called *bottom*, expressing the notion of *total like* or *perfect* assignment. If a tuple receives such valuation from a function, it should not increase its overall valuation. Consequently, $\bot$ must be the *unit* element of $\otimes$ ($\forall\, a \in A, a \otimes \bot = a$).
- The operator $\oplus$ is used to select valuations from different entries of the same function (i.e. different assignments of the same variables).
  - There is no implicit order over the different assignments of the variables. This means that the $\oplus$ operator must also be *associative* and *commutative*.
  - Since the $\oplus$ operator is used for choosing the *best* alternative, it must work in coherence with the order $\leq_{\mathcal{K}}$.

There are two approaches to incorporate the previous list of requirements: *Valued CSP* (VCSP) [129] and *Semiring CSP* (SCSP) [17].

- In the *Valued CSP framework* the valuation structure $\mathcal{K} = (A, \otimes, \oplus)$ is required to satisfy the following conditions:

  - A must be a totally ordered set with $\top$ and $\bot$ being the highest and lowest values, respectively.
  - The $\otimes$ operator must be commutative and associative.
  - The $\otimes$ operator must satisfy that $\forall\, a, b, c \in A \;\; a \leq_{\mathcal{K}} b \Rightarrow a \otimes c \leq_{\mathcal{K}} b \otimes c$.
  - The $\bot$ element is unit with respect $\otimes$.
  - The $\top$ element is absorbing with respect $\otimes$.
  - The $\oplus$ operator is the *min* with respect to the total order.

  A *Valued CSP problem* is a reasoning task such that its valuation structure satisfies the previous conditions.

- In the *Semiring CSP framework* the valuation structure $\mathcal{K} = (A, \otimes, \oplus)$ is required to satisfy that:

  - A is an arbitrary set containing $\top$ and $\bot$.
  - $\otimes$ is commutative and associative.
  - The $\bot$ element is unit with respect $\otimes$.
  - The $\top$ element is absorbing with respect $\otimes$.
  - $\oplus$ is commutative and associative.
  - $\oplus$ is idempotent.
  - $\otimes$ distributes over $\oplus$.

Task	CSP	WCSP	#CSP	MAP	MPE
$\mathcal{K}$	$(\{true, false\}, \wedge, \vee)$	$(\mathbb{N}^{\infty}, +, \min)$	$(\{1, 0\}, \times, +)$	$([0, 1], \times, +)$	$([0, 1], \times, \max)$
VCSP	√	√			√
SCSP	√	√			√
SS	√	√	√	√	√

Figure 4.1: Relation between reasoning tasks and algebraic frameworks.

– The order over A is defined as[2] $a \leq_{\mathcal{K}} b$ iff $a \oplus b = a$.

Valuation structures that satisfy the previous conditions are called *c-semirings* because they are semirings with some additional requirements. A *Semiring CSP problem* is a reasoning task such that its valuation structure is a c-semiring.

It is easy to see that the SCSP framework is strictly more general than the VCSP framework [17]. The essential difference is that SCSP gives more freedom to the $\oplus$ operator which allows A to be partially ordered. In multi-objective optimization problems, it is clear that we need to deal with partially ordered valuations.

At this point it may be worth to mention the existence of another framework called *Shenoy Shaffer* (S-S) [133]. Initially, its algebraic structure, called *valuation algebra*, has been described by means of three axioms and related to inference algorithms. It has been further extended and studied in detail in [134, 81, 80]. Mainly, it verifies the same properties as the SCSP framework except for the idempotency of the $\oplus$ operator. This absence allows the S-S framework to capture not only optimization, but also counting tasks. This framework is specially appropriate for modeling bayesian networks, where both types of tasks are posed. Since we are only interested in optimization problems, we do not consider the S-S framework in our work.

[2]In the original formulation of SCSP optimization was assumed to be maximization. Therefore, a higher value of the semiring was interpreted as a better value. For coherence, $\leq_{\mathcal{K}}$ was defined as $a \leq_{\mathcal{K}} b$ iff $a + b = b$. Since we are dealing with minimization tasks we reverse the semantics of the order and interpret small values as better.

Table 4.1 relates algebraic frameworks and graphical model tasks by telling the expressive power of each framework. The first and second rows contain the main reasoning tasks on graphical models and their associated valuation structures, respectively. The following three rows tell, for each algebraic framework (i.e., VCSP, SCSP, S-S), if it can be used to model the corresponding reasoning task. As it can be seen, S-S is the only framework that can model counting tasks. Regarding VCSP and SCSP, we see that the two of them can model the most usual optimization tasks. However, as we show in the rest of this chapter, SCSP can model multi-objective optimization while VCSP cannot. The obvious reason is that VCSP cannot deal with partially ordered valuation sets. This is true by definition of VCSP.

4.2 Important properties of c-semirings

In the following, we list a number of properties that any c-semiring $\mathcal{K} = (A, \otimes, \oplus)$ accomplishes. We only refer to those theorems that will be useful thoughout the Thesis.

Theorem 4.2.1 *$\leq_{\mathcal{K}}$ is a partial order (i.e., reflexive, transitive and antysimmetric).*

Proof Theorem 2.3 in [17].

Theorem 4.2.2 *$\otimes$ and $\oplus$ are monotone over $\leq_{\mathcal{K}}$. Formally,*

$$\forall\, a, b, c \in A \text{ if } a \leq_{\mathcal{K}} b \text{ then } a \otimes c \leq_{\mathcal{K}} b \otimes c \text{ and } a \oplus c \leq_{\mathcal{K}} b \oplus c$$

Proof Theorem 2.4 in [17].

Theorem 4.2.3 *Let $a, b, c, d \in A$. If $a \leq_{\mathcal{K}} b$ and $c \leq_{\mathcal{K}} d$ then $a \otimes c \leq_{\mathcal{K}} b \otimes d$.*

Proof By monotonicity of $\leq_{\mathcal{K}}$, $a \otimes c \leq_{\mathcal{K}} b \otimes c$ and $c \otimes b \leq_{\mathcal{K}} d \otimes b$. Then, the theorem holds by commutativity of $\otimes$ and transitivity of $\leq_{\mathcal{K}}$.

Theorem 4.2.4 *$\otimes$ is intensive, that is, $\forall\, a, b \in A, a \leq_{\mathcal{K}} a \otimes b$.*

Proof Theorem 2.5 in [17].

Theorem 4.2.5 *Let P be a reasoning task defined over the c-semiring $\mathcal{K}$. Let $\mathcal{F}$ be its set of functions. Let $\mathcal{F}'$ be a subset of $\mathcal{F}$ and P' the reasoning task defined over this subset. If the optimal valuation of $\mathcal{P}$ is α, then the optimal valuation of $\mathcal{P}'$ is β, with $\beta \leq_{\mathcal{K}} \alpha$, where $\leq_{\mathcal{K}}$ is the partial order defined by the c-semiring $\mathcal{K}$.*

Proof Theorem 3.18 in [17].

4.3 Semiring-based multi-objective optimization

Consider a multi-objective problem P_{mo} with multiobjective function $F = (F_1, \ldots, F_p)$. Recall that we are concerned with the computation of its efficient frontier $\mathcal{E}(P_{mo}) = \{F(X) \mid \forall\, X'\ F(X') \not< F(X)\}$, where $<$ is the domination partial order among vectors (see Definition 2.2.1).

We assume that the independent optimization of each objective F_j can be modeled as an optimization task $P_j = (\mathcal{X}, \mathcal{D}, A_j, \mathcal{F}_j, \bigotimes_j, \bigoplus_j)$ such that $\mathcal{K}_j = (A_j, \otimes_j, \oplus_j)$ is a c-semiring with top element $\top_j$ and bottom element $\bot_j$. The objective function is,

$$F_j(\mathcal{X}) = \bigotimes_{f \in \mathcal{F}}{}_j\, f$$

and its optimum is,

$$\bigoplus_{\mathcal{X}}{}_j\, F_j(\mathcal{X})$$

The set of variables $\mathcal{X}$ and their domain values $\mathcal{D}$ is common to all the objectives.

4.3.1 p-composition SCSP

p-Dimensional c-semiring

The composition of p c-semirings $\mathcal{K}_j$ is a new c-semiring $\mathcal{K}_\mathcal{C}$ that combines them in a cartesian product manner.

Definition 4.3.1 *Given p c-semirings $\mathcal{K}_j = (A_j, \oplus_j, \otimes_j)$, for $j = 1, \ldots, p$, their* composition *is $\mathcal{K}_\mathcal{C} = (A_\mathcal{C}, \oplus_\mathcal{C}, \otimes_\mathcal{C})$ where:*

- $A_\mathcal{C} = A_1 \times A_2 \times \ldots \times A_p$
- $(a_1, \ldots, a_p) \oplus_\mathcal{C} (b_1, \ldots, b_p) = (a_1 \oplus_1 b_1, \ldots, a_p \oplus_p b_p)$
- $(a_1, \ldots, a_p) \otimes_\mathcal{C} (b_1, \ldots, b_p) = (a_1 \otimes_1 b_1, \ldots, a_p \otimes_p b_p)$

Theorem 4.3.1 *[17] $\mathcal{K}_\mathcal{C}$ is a c-semiring whose top and bottom elements are $(\top_1, \ldots, \top_p)$ and $(\bot_1, \ldots, \bot_p)$, respectively.*

Property 4.3.1 *The order $\leq_{\mathcal{K}_\mathcal{C}}$ induced by $\oplus_\mathcal{C}$ is,*

$$(a_1, \ldots, a_p) \leq_{\mathcal{K}_\mathcal{C}} (b_1, \ldots, b_p) \text{ iff } \underset{1 \leq j \leq p}{\forall} a_j \leq_{\mathcal{K}_j} b_j$$

In general, $\leq_{\mathcal{K}_\mathcal{C}}$ is a partial order, even if each of the $\leq_{\mathcal{K}_j}$ is a total order. Observe that $\leq_{\mathcal{K}_\mathcal{C}}$ coincides with the partial order used in multi-objective optimization (see Definition 2.2.1). For that reason, some authors have claimed that the p-composition of the p semirings associated to the p objectives can be used to model and solve multi-objective optimization problems [18, 19]. Now we develop this idea and show that it cannot be considered fully satisfactory because their notion of solution is only very loosely related to the concept of pareto-optimality.

p-composition SCSP Problems

The p-composition of p SCSP problems is a new SCSP problem over the p-composition of their c-semirings.

Definition 4.3.2 *Given p SCSP problems* $P_j = (\mathcal{X}, \mathcal{D}, A_j, \mathcal{F}_j, \bigotimes_j, \bigoplus_j)$*, their p-composition is a new SCSP problem* $P_{\mathcal{C}} = (\mathcal{X}, \mathcal{D}, A_{\mathcal{C}}, \mathcal{F}_{\mathcal{C}}, \bigotimes_{\mathcal{C}}, \bigoplus_{\mathcal{C}})$*.* $P_{\mathcal{C}}$ *uses c-semiring* $\mathcal{K}_{\mathcal{C}} = (A_{\mathcal{C}}, \otimes_{\mathcal{C}}, \oplus_{\mathcal{C}})$ *.*

Each function $f : l(var(f)) \rightarrow A_j$ *belonging to each* $\mathcal{F}_j$ *is transformed into a new function* $f' : l(var(f)) \rightarrow A_{\mathcal{C}}$ *defined as*

$$f'(t) = (\bot_1, \ldots, \bot_{j-1}, f(t), \bot_{j+1} \ldots, \bot_p)$$

In words, function outcomes of f are transformed to an equivalent vectorial representation in f'*. Then, the set* $\mathcal{F}_{\mathcal{C}}$ *contains the new* f' *functions.*

Note that the optimization problem $P_{\mathcal{C}}$ does not return a set of vectors, but a single cost vector $\vec{v}$. Therefore, in general $P_{\mathcal{C}}$ does not compute the efficient frontier of P_{mo}. Next, we show that the j^{th} component of $\vec{v}$ is the optimum of problem P_j, that is, the optimum of the j^{th} objective function of P_{mo} when its p objective functions are considered independently. As a consequence, it may not even exist any complete assignment X such that $F(X) = \vec{v}$. The following proposition characterizes the optimum of $P_{\mathcal{C}}$ in terms of the original multi-objective problem P_{mo}.

Property 4.3.2 *Consider a multi-objective problem* P_{mo}*. Let* $\mathcal{E}(P_{mo})$ *denote its efficient frontier. Let* $P_{\mathcal{C}}$ *be the p-composition of the p objectives of* P_{mo} *and* $\vec{v}$ *be its optimum. Namely,*

$$\bigoplus_{\mathcal{X}}{}_{\mathcal{C}}(\bigotimes_{f \in \mathcal{F}_{\mathcal{C}}}{}_{\mathcal{C}} f) = \vec{v}$$

Then, it can be shown that $\vec{v}$ *is the greatest vector that is lower or equal to every element in* $\mathcal{E}(P_{mo})$*.*

Formally, $\vec{v}$ *is the greatest lower bound of* $\mathcal{E}(P_{mo})$ *(i.e.,* $\vec{v} = glb(\mathcal{E}(P_{mo}))$*).*

A straightforward consequence of the previous proposition is that $P_{\mathcal{C}}$ computes the efficient frontier of the original problem P_{mo} if and only if the efficient frontier is a singleton. Otherwise, $P_{\mathcal{C}}$ only computes the best singleton underapproximation.

The previous results are illustrated with the following example.

Example 4.3.1 *Consider a bi-objective optimization problem P_{mo}. Let $\mathcal{X} = \{x_1, x_2\}$ and let $D_1 = D_2 = \{0, 1\}$. The first objective function is $F_1 = f_1 + f_2$ where $f_1(x_1) = x_1$ and $f_2(x_2) = x_2$. The second objective function is $F_2 = h_1 + h_2$ where $h_1(x_1) = 1 - x_1$ and $h_2(x_2) = x_2$. The multi-objective function is $F = (F_1, F_2)$. It is easy to see that the efficient frontier of P_{mo} is $\mathcal{E}(P_{mo}) = \{(1, 0), (0, 1)\}$. Note that cost vectors $(1, 2)$ and $(2, 1)$ are not efficient because they are dominated by the cost vectors in $\mathcal{E}(P_{mo})$.*

Let us consider the SCSP problem $P_{\mathcal{C}}$. First, the independent optimization of each objective function F_j can be modeled as a WCSP problem

$$P_j = (\{x_1, x_2\}, \{0, 1\}, \mathbb{N}^\infty, \mathcal{F}_j, +, \min)$$

over c-semiring

$$\mathcal{K}_j = \langle \mathbb{N}^\infty, +, \min \rangle$$

with $\top_j = \infty$ and $\bot_j = 0$. The set of functions for the first and second WCSPs are $\mathcal{F}_1 = \{f_1, f_2\}$ and $\mathcal{F}_2 = \{h_1, h_2\}$, respectively.

The 2-composition of c-semirings $\mathcal{K}_1$ and $\mathcal{K}_2$ is

$$\mathcal{K}_{\mathcal{C}} = \langle \overrightarrow{A}, \overrightarrow{+}, \overrightarrow{\min} \rangle$$

where,

- $\overrightarrow{A} = \mathbb{N}^\infty \times \mathbb{N}^\infty$
- $\overrightarrow{\min} = (\min, \min)$
- $\overrightarrow{+} = (+, +)$
- $\top = (\infty, \infty)$
- $\bot = (0, 0)$

Then, the SCSP problem defined over $\mathcal{K}_{\mathcal{C}}$ is

$$P_{\mathcal{C}} = (\{x_1, x_2\}, \{0, 1\}, \overrightarrow{A}, \mathcal{F}_{\mathcal{C}}, \overrightarrow{+}, \overrightarrow{\min})$$

The set of functions is $\mathcal{F}_{\mathcal{C}} = \{f'_1, f'_2, h'_1, h'_2\}$ where

$$f'_1(x_1) = (x_1, 0) \qquad f'_2(x_2) = (x_2, 0)$$
$$h'_1(x_1) = (0, 1 - x_1) \qquad h'_2(x_2) = (0, x_2)$$

Note that the set of functions in $\mathcal{F}_\mathcal{C}$ represents the same information as the corresponding functions in $\mathcal{F}_1$ and $\mathcal{F}_2$ but extended to the vectorial context of $P_\mathcal{C}$. The optimization problem $P_\mathcal{C}$ computes,

$$\overrightarrow{\min_{\mathcal{X}}}\{\overrightarrow{\sum_{f \in \mathcal{F}_\mathcal{C}}} f\}$$

which is equivalent to,

$$(\min_{\mathcal{X}}(F_1), \min_{\mathcal{X}}(F_2))$$

The result of the previous expression is the vector $(0, 0)$, which is not the efficient frontier $\mathcal{E}(P_{mo})$. It is easy to see that 0 is the optimum of both optimization problems P_1 and P_2 when considered independently.

4.3.2 Frontier SCSP

We have shown that solving the p-composition of p objective functions does not capture the efficient frontier of a multi-objective problem P_{mo}. The main reason is that its valuations are single cost vectors while the efficient frontier is a set of cost vectors.

In the following, we show how to build from a generic c-semiring $\mathcal{K}$ a new c-semiring $\mathcal{L}(\mathcal{K})$ whose elements are not-empty sets of elements of $\mathcal{K}$. Then, we demonstrate that if a SCSP problem P over c-semiring $\mathcal{K}$ is rephrased over $\mathcal{L}(\mathcal{K})$, its optimum is the set of optima of P.

Frontier algebra

Consider a c-semiring $\mathcal{K} = (A, \otimes, \oplus)$, where $\leq_\mathcal{K}$ is the (possibly partial) order of A. We introduce the set of non-dominated elements of a subset $S \subseteq A$.

Definition 4.3.3 *Let S be a subset of A. The* set of non-dominated elements *of S is,*

$$\|S\| = \{a \in S \mid \forall\, b \in S, b \not<_\mathcal{K} a\}$$

The idea of the set of non-dominated elements is to keep only the *best* valuations in S according to the partial order. We say that a set S is *a set of non-dominated elements* if it does not contain dominated elements (i.e. $S = \|S\|$). We adopt the usual multi-objective terminology where a set of non-dominated elements is called a *frontier*.

Definition 4.3.4 *Let A be a set of valuations. The* frontier space *of A, noted $\mathcal{L}(A)$, is the set of subsets of A that do not contain dominated elements. Formally, $\mathcal{L}(A) = \{S \subseteq A \mid S = \|S\|\}$.*

The *frontier algebra* of $\mathcal{K}$ is a new c-semiring $\mathcal{L}(\mathcal{K})$ whose set of valuations is the frontier space of A.

Definition 4.3.5 *Let $\mathcal{K} = (A, \otimes, \oplus)$ be a c-semiring. Then, its* frontier algebra *is*

$$\mathcal{L}(\mathcal{K}) = (\mathcal{L}(A),\ \otimes_{\mathcal{L}},\ \oplus_{\mathcal{L}})$$

where,

- $\mathcal{L}(A)$ *is the frontier space of A*
- $S \otimes_{\mathcal{L}} T = \|\{a \otimes b \mid a \in S, b \in T\}\|$
- $S \oplus_{\mathcal{L}} T = \|S \cup T\|$

Theorem 4.3.2 *Let $\mathcal{K}$ be a c-semiring with top element $\top$ and bottom element $\bot$. Then, its frontier algebra $\mathcal{L}(\mathcal{K})$ is a c-semiring whose top and bottom elements are $\{\top\}$ and $\{\bot\}$, respectively.*

Proof Let S, T, R be arbitrary elements of $\mathcal{L}(A)$. We proof, one by one, the required conditions.

- commutativity of $\otimes_{\mathcal{L}}$. By definition, $S \otimes_{\mathcal{L}} T = \|\{a \otimes b \mid a \in S, b \in T\}\|$. Since $\otimes$ is commutative, the previous expression is equal to $\|\{b \otimes a \mid b \in T, a \in S\}\| = T \otimes_{\mathcal{L}} S$.

- associativity of $\otimes_{\mathcal{L}}$. We have to proof that $(S \otimes_{\mathcal{L}} T) \otimes_{\mathcal{L}} R = S \otimes_{\mathcal{L}} (T \otimes_{\mathcal{L}} R)$. Suppose that the previous equality does not hold. Then, it would imply that:

  i. there may exist an element $a \in (S \otimes_{\mathcal{L}} T) \otimes_{\mathcal{L}} R$, such that $a \notin S \otimes_{\mathcal{L}} (T \otimes_{\mathcal{L}} R)$; or,

  ii. there may exist an element $a \in S \otimes_{\mathcal{L}} (T \otimes_{\mathcal{L}} R)$, such that $a \notin (S \otimes_{\mathcal{L}} T) \otimes_{\mathcal{L}} R$.

  We show that both cases are impossible.

  Consider the first case. Since $a \notin S \otimes_{\mathcal{L}} (T \otimes_{\mathcal{L}} R)$, it means that there exist an element $a' \in S \otimes_{\mathcal{L}} (T \otimes_{\mathcal{L}} R)$ such that $a' <_{\mathcal{K}} a$. Element a comes from the combination of three elements $a = (s \otimes t) \otimes r$ where $s \in S$, $t \in T$ and $r \in R$. Element a' comes from the combination of three elements $a' = s' \otimes (t' \otimes r')$ where $s' \in S$, $t' \in T$ and $r' \in R$. By associativity of operator $\otimes$, $a' = (s' \otimes t') \otimes r'$. Then, either $s' \otimes t' \in S \otimes_{\mathcal{L}} T$ or $s' \otimes t' \notin S \otimes_{\mathcal{L}} T$:

  - If $s' \otimes t' \in S \otimes_{\mathcal{L}} T$, since $a' <_{\mathcal{K}} a$ and by definition of $\otimes_{\mathcal{L}}$, $a \notin (S \otimes_{\mathcal{L}} T) \otimes_{\mathcal{L}} R$, which contradicts the hypothesis.
  - If $s' \otimes t' \notin S \otimes_{\mathcal{L}} T$, by definition of $\otimes_{\mathcal{L}}$, there exists an element $s'' \in S$, $t'' \in T$ such that $s'' \otimes t'' \in S \otimes_{\mathcal{L}} T$ and $s'' \otimes t'' <_{\mathcal{K}} s' \otimes t'$. By monotonicity of $<_{\mathcal{K}}$, $(s'' \otimes t'') \otimes r' <_{\mathcal{K}} (s' \otimes t') \otimes r'$. By transitivity of $<_{\mathcal{K}}$, $(s'' \otimes t'') \otimes r' <_{\mathcal{K}} (s \otimes t) \otimes r$. Then, by definition of $\otimes_{\mathcal{L}}$, $a \notin (S \otimes_{\mathcal{L}} T) \otimes_{\mathcal{L}} R$, which contradicts the hypothesis.

  The proof for the second case is the same as above, but interchanging the role of a and a', and S and R.

- $\{\perp\}$ is the identity element of $\otimes_{\mathcal{L}}$. We have to proof that $\{\perp\} \otimes_{\mathcal{L}} S = S$. By definition, $\{\perp\} \otimes_{\mathcal{L}} S = \{\perp \otimes a \mid a \in S\}$. Since $\perp$ is the identity of $\otimes$, the previous expression is equal to $\{a \mid a \in S\} = S$.

- $\{\top\}$ is the absorbing element of $\otimes_{\mathcal{L}}$. We have to proof that $\{\top\} \otimes_{\mathcal{L}} S = \{\top\}$. By definition, $\{\top\} \otimes_{\mathcal{L}} S = \{\top \otimes a \mid a \in S\}$. Since $\top$ is the absorbing element of $\otimes$, the previous expression is equal to $\{\top\}$.

- commutativity of $\oplus_{\mathcal{L}}$. By definition, $S \oplus_{\mathcal{L}} T = \|S \cup T\|$. Since set union is commutative, $\|S \cup T\| = \|T \cup S\|$ which is by definition $T \oplus_{\mathcal{L}} S$.

- associativity of $\oplus_{\mathcal{L}}$. By definition, $(S \oplus_{\mathcal{L}} T) \oplus_{\mathcal{L}} R = \|\|S \cup T\| \cup R\|$ and $S \oplus_{\mathcal{L}} (T \oplus_{\mathcal{L}} R) = \|S \cup \|T \cup R\|\|$. Clearly, the two expressions are equivalent to $\|S \cup T \cup R\|$.

- idempotency of $\oplus_{\mathcal{L}}$. We have to proof that $S \oplus_{\mathcal{L}} S = S$. By definition, $S \oplus_{\mathcal{L}} S = \|S \cup S\|$. Since set union does not allow repeted elements, the previous expression is equal to $\|S\| = S$.

- $\otimes_{\mathcal{L}}$ distributes over $\oplus_{\mathcal{L}}$. We have to proof that $S \otimes_{\mathcal{L}} (T \oplus_{\mathcal{L}} R) = (S \otimes_{\mathcal{L}} T) \oplus_{\mathcal{L}} (S \otimes_{\mathcal{L}} R)$. Suppose that the previous equality does not hold. Then, it would imply that:

  i. there exists an element $a \in A$ such that $a \in S \otimes_{\mathcal{L}} (T \oplus_{\mathcal{L}} R)$ but $a \notin (S \otimes_{\mathcal{L}} T) \oplus_{\mathcal{L}} (S \otimes_{\mathcal{L}} R)$; or

  ii. there exists an element $a \in A$ such that $a \in (S \otimes_{\mathcal{L}} T) \oplus_{\mathcal{L}} (S \otimes_{\mathcal{L}} R)$ but $a \notin S \otimes_{\mathcal{L}} (T \oplus_{\mathcal{L}} R)$.

  We show that both cases are not possible.

  Consider the first case. Since $a \notin (S \otimes_{\mathcal{L}} T) \oplus_{\mathcal{L}} (S \otimes_{\mathcal{L}} R)$, it means that there exists an element $a' \in A$ such that $a' \in (S \otimes_{\mathcal{L}} T) \oplus_{\mathcal{L}} (S \otimes_{\mathcal{L}} R)$ and $a' <_{\mathcal{K}} a$. Element a' comes from the combination of two elements $s', u' \in A$ such that $a' = s' \otimes u'$ where $s' \in S$, and $u' \in T$ or $u' \in R$. Element u' can either belong to $T \oplus_{\mathcal{L}} R$ or not:

  - If $u' \in T \oplus_{\mathcal{L}} R$, since $s' \otimes u' <_{\mathcal{K}} a$ and by definition of $\otimes_{\mathcal{L}}$, $a \notin S \otimes_{\mathcal{L}} (T \oplus_{\mathcal{L}} R)$, which contradicts the hypothesis.

- If $u' \notin T \oplus_{\mathcal{L}} R$, by definition of $\oplus_{\mathcal{L}}$, there exists an element $w' \in T \oplus_{\mathcal{L}} R$ such that $w' <_{\mathcal{K}} u'$. By monotonicity of $<_{\mathcal{K}}$, $s' \otimes w' <_{\mathcal{K}} s' \otimes u'$. By transitivity of $<_{\mathcal{K}}$, $s' \otimes_{\mathcal{K}} w' <_{\mathcal{K}} a$. Then, by definition of $\otimes_{\mathcal{L}}$, $a \notin S \otimes_{\mathcal{L}} (T \oplus_{\mathcal{L}} R)$, which contradicts the hypothesis.

Consider the second case. Since $a \notin S \otimes_{\mathcal{L}} (T \oplus_{\mathcal{L}} R)$, it means that there exists an element $a' \in A$ such that $a' \in S \otimes_{\mathcal{L}} (T \oplus_{\mathcal{L}} R)$ and $a' <_{\mathcal{K}} a$. Moreover, since $a \in (S \otimes_{\mathcal{L}} T) \oplus_{\mathcal{L}} (S \otimes_{\mathcal{L}} R)$, then either $a \in S \otimes_{\mathcal{L}} T$ or $a \in S \otimes_{\mathcal{L}} R$. Element a' comes from the combination of two elements $s', u' \in A$ such that $a' = s' \otimes u'$ where $s' \in S$ and $u' \in T \oplus_{\mathcal{L}} R$. As a consequence, either $u' \in T$ or $u' \in R$.

If $u' \in T$, then either $a' \in S \otimes_{\mathcal{L}} T$ or $a' \notin S \otimes_{\mathcal{L}} T$:

- If $a' \in S \otimes_{\mathcal{L}} T$, then either by definition of $\otimes_{\mathcal{L}}$ when $a \in S \otimes_{\mathcal{L}} T$ or by definition of $\oplus_{\mathcal{L}}$ when $a \in S \otimes_{\mathcal{L}} R$, $a \notin (S \otimes_{\mathcal{L}} T) \oplus_{\mathcal{L}} (S \otimes_{\mathcal{L}} R)$, which contradicts the hypothesis.
- If $a' \notin S \otimes_{\mathcal{L}} T$, then there exists an element $a'' \in A$ such that $a'' \in S \otimes_{\mathcal{L}} T$ and $a'' <_{\mathcal{K}} a'$. By transitivity of $<_{\mathcal{K}}$, $a'' <_{\mathcal{K}} a$. As a consequence, either by definition of $\otimes_{\mathcal{L}}$ when $a \in S \otimes_{\mathcal{L}} T$ or by definition of $\oplus_{\mathcal{L}}$ when $a \in S \otimes_{\mathcal{L}} R$, $a \notin (S \otimes_{\mathcal{L}} T) \oplus_{\mathcal{L}} (S \otimes_{\mathcal{L}} R)$, which contradicts the hypothesis.

When $u' \in R$, the reasoning is the same as above but interchanging the role of T and R.

It is worthwhile to see the ordering $\leq_{\mathcal{L}(\mathcal{K})}$ that the SCSP approach associates to the frontier algebra $\mathcal{L}(\mathcal{K})$, because it will be pivotal in multi-objective branch and bound (Chapter 5).

Property 4.3.3 *Let $S, T \in \mathcal{L}(A)$. Then, $S \leq_{\mathcal{L}(\mathcal{K})} T$ iff for each $b \in T$ there exists $a \in S$ such that $a \leq_{\mathcal{K}} b$.*

Proof By definition, $S \leq_{\mathcal{L}(\mathcal{K})} T$ iff $S \oplus_{\mathcal{L}} T = S$. Clearly, $S \oplus_{\mathcal{L}} T = S$ iff every element of T is dominated by some element of S, which corresponds to the statement of the proposition.

Property 4.3.4 *Let $S, T \subseteq A$. If $S \subseteq T$ then $\|T\| \leq_{\mathcal{L}(\mathcal{K})} \|S\|$.*

Proof We proceed by contradiction. Suppose that $\|T\| \not\leq_{\mathcal{L}(\mathcal{K})} \|S\|$. This means that $\exists\, s \in \|S\|$ such that $s \notin \|T\|$ and $\forall\, t \in \|T\|, t \not\leq_{\mathcal{K}} s$. Since $S \subseteq T$, $s \notin \|T\|$ and by definition of $\|T\|$, $\exists\, t' \in \|T\|$ such that $t' \leq_{\mathcal{K}} s$, which contradicts the hypothesis.

Moreover, since $\mathcal{L}(\mathcal{K})$ is proved to be a c-semiring, $\leq_{\mathcal{L}(\mathcal{K})}$ is a partial order, operators $\otimes_{\mathcal{L}}$ and $\oplus_{\mathcal{L}}$ are monotone with respect to $\leq_{\mathcal{L}(\mathcal{K})}$, and $\otimes_{\mathcal{L}}$ is intensive.

Frontier SCSP Problems

The frontier extension of a SCSP problem P is a new SCSP problem $\mathcal{L}(P)$ that, in a way, is able to consider different incomparable optimal alternatives of P.

Definition 4.3.6 *Let $P = (\mathcal{X}, \mathcal{D}, A, \mathcal{F}, \bigotimes, \bigoplus)$ be a SCSP problem defined over a c-semiring $\mathcal{K} = (A, \otimes, \oplus)$. Its frontier extension is a new SCSP problem $\mathcal{L}(P) = (\mathcal{X}, \mathcal{D}, \mathcal{L}(A), \mathcal{L}(\mathcal{F}), \bigotimes_{\mathcal{L}}, \bigoplus_{\mathcal{L}})$ over the frontier c-semiring $\mathcal{L}(\mathcal{K}) = (\mathcal{L}(A), \otimes_{\mathcal{L}}, \oplus_{\mathcal{L}})$.*

Each function $f : l(var(f)) \to A$ is trivially transformed into a new function $f' : l(var(f)) \to \mathcal{L}(A)$ defined as $f'(t) = \{f(t)\}$. In words, function outcomes of f are transformed to singleton sets in f'. Then, the set $\mathcal{L}(\mathcal{F})$ contains the new f' functions.

The following theorem shows that the optimum of $\mathcal{L}(P)$ corresponds to the set of valuations associated with the optimal solutions in P.

Theorem 4.3.3 *Consider a SCSP problem* $P = (\mathcal{X}, \mathcal{D}, A, \mathcal{F}, \bigotimes, \bigoplus)$ *defined over a (possibly partially ordered) c-semiring* $\mathcal{K}$. *Let* $F(\mathcal{X}) = \bigotimes_{f \in \mathcal{F}} f$ *be its objective function. Let* $\mathcal{L}(P) = (\mathcal{X}, \mathcal{D}, \mathcal{L}(A), \mathcal{L}(\mathcal{F}), \bigotimes_{\mathcal{L}}, \bigoplus_{\mathcal{L}})$ *be the frontier extension of* P *and let*

$$F_{\mathcal{L}}(\mathcal{X}) = \bigotimes_{\substack{\mathcal{L} \\ f \in \mathcal{L}(\mathcal{F})}} f$$

be its objective function. The optimization task $\mathcal{L}(P)$ *computes the set of optimal solutions of* P. *Formally,*

$$\bigoplus_{\substack{\mathcal{L} \\ \mathcal{X}}} F_{\mathcal{L}}(\mathcal{X}) = \{F(X) \mid \forall\, X', F(X') \not\leq_{\mathcal{K}} F(X)\}$$

Proof By definition of $\bigoplus_{\mathcal{L}}$,

$$\bigoplus_{\substack{\mathcal{L} \\ \mathcal{X}}} F_{\mathcal{L}}(\mathcal{X}) = \| \bigcup_{X \in l(var(\mathcal{X}))} F_{\mathcal{L}}(X) \|$$

By definition of $F_{\mathcal{L}}$,

$$\bigoplus_{\substack{\mathcal{L} \\ \mathcal{X}}} F_{\mathcal{L}}(\mathcal{X}) = \| \bigcup_{X \in l(var(\mathcal{X}))} (\bigotimes_{\substack{\mathcal{L} \\ f \in \mathcal{L}(\mathcal{F})}} f(X)) \|$$

By definition of $\mathcal{L}(\mathcal{F})$,

$$\bigoplus_{\substack{\mathcal{L} \\ \mathcal{X}}} F_{\mathcal{L}}(\mathcal{X}) = \| \bigcup_{X \in l(var(\mathcal{X}))} (\bigotimes_{\substack{\mathcal{L} \\ f \in \mathcal{F}}} \{f(X)\}) \|$$

Since all $\{f(X)\}$ are singletons, then $\{f(X)\} \otimes_{\mathcal{L}} \{g(X)\} = \{f(X) \otimes g(X)\}$. Then,

$$\bigoplus_{\substack{\mathcal{L} \\ \mathcal{X}}} F_{\mathcal{L}}(\mathcal{X}) = \| \bigcup_{X \in l(var(\mathcal{X}))} \{\bigotimes_{f \in \mathcal{F}} f(X)\} \|$$

By definition of F,

$$\bigoplus_{\substack{\mathcal{L} \\ \mathcal{X}}} F_{\mathcal{L}}(\mathcal{X}) = \| \bigcup_{X \in l(var(\mathcal{X}))} \{F(X)\} \|$$

By definition of the set union,

$$\bigoplus_{\mathcal{X}}{}_{\mathcal{L}} F_{\mathcal{L}}(\mathcal{X}) = \|\{F(X) \mid X \in l(var(\mathcal{X}))\}\|$$

By definition of the set of non-dominated elements,

$$\bigoplus_{\mathcal{X}}{}_{\mathcal{L}} F_{\mathcal{L}}(\mathcal{X}) = \{F(X) \mid \forall\, X', F(X') \not\leq_{\mathcal{K}} F(X)\}$$

4.3.3 Multi-objective SCSP

Finally, we can put all the pieces together and show how to construct from a multiobjective problem P_{mo} a SCSP problem whose optimization task computes the efficient frontier of P_{mo}.

Let $\mathcal{K}_1, \ldots, \mathcal{K}_p$ be the c-semirings of the objectives of P_{mo}. Let $P_1, \ldots, P_p$ be the optimization problems of objective functions $F_1, \ldots, F_p$ when considered independently. Let $\mathcal{K}_{\mathcal{C}}$ denote the p-composition of the p c-semirings $\mathcal{K}_j$ and let $P_{\mathcal{C}}$ be the p-composition of the p optimization problems. Let $F_{\mathcal{C}}$ be the objective function of $P_{\mathcal{C}}$. Therefore, $F_{\mathcal{C}} = (F_1, \ldots, F_p)$. As a consequence, the set of optimal complete assignments of $P_{\mathcal{C}}$ corresponds to the set of efficient solutions of P_{mo}.

Now, let $\mathcal{L}(\mathcal{K}_{\mathcal{C}})$ be the frontier algebra of $\mathcal{K}_{\mathcal{C}}$ and let $\mathcal{L}(P_{\mathcal{C}})$ be the frontier extension of $P_{\mathcal{C}}$. From Theorem 4.3.3, the optimization problem $\mathcal{L}(P_{\mathcal{C}})$ computes the set of valuations associated with incomparable optimal solutions of $P_{\mathcal{C}}$. Since this set of optimal solutions corresponds to the set of efficient solutions of P_{mo}, a direct consequence is that $\mathcal{L}(P_{\mathcal{C}})$ computes the efficient frontier of P_{mo}.

The following property and theorem will be useful in future Chapters.

Property 4.3.5 *Let $P = (\mathcal{X}, \mathcal{D}, A, \mathcal{F}, \bigotimes_{\mathcal{L}}, \bigoplus_{\mathcal{L}})$ and $P = (\mathcal{X}, \mathcal{D}, A, \mathcal{F}', \bigotimes_{\mathcal{L}}, \bigoplus_{\mathcal{L}})$ be two multi-objective SCSP problems such that $\mathcal{F}' \subseteq \mathcal{F}$. Then $\mathcal{E}(P') \leq_{\mathcal{L}(\mathcal{K}_{\mathcal{C}})} \mathcal{E}(P)$.*

Proof This property is the straighforward consequence of Theorem 4.2.5 and the fact that $\mathcal{L}(\mathcal{K}_{\mathcal{C}})$ is a c-semiring.

Theorem 4.3.4 *Let $P = (\mathcal{X}, \mathcal{D}, A, \mathcal{F}, \bigotimes_{\mathcal{L}}, \bigoplus_{\mathcal{L}})$ be a multi-objective SCSP problem. Let $(\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k)$ be a partition of $\mathcal{F}$ and $(P_1, P_2, \ldots, P_k)$ be the problems induced by each partition. Then,*

$$\sum_{i=1}^{k}{}^{mo} \mathcal{E}(P_i) \leq_{\mathcal{L}(\mathcal{K}_\mathcal{C})} \mathcal{E}(P)$$

Proof Let $\vec{u} \in \mathcal{E}(P)$ and let $X(\vec{u})$ be the complete assignment such that $F(X(\vec{u})) = \sum F_i(X(\vec{u})) = \vec{u}$. Clearly, $\forall\, i = 1, \ldots, k\, \mathcal{E}(P_i) \leq_{\mathcal{L}(\mathcal{K}_\mathcal{C})} \{F_i(X(\vec{u}))\}$. By Theorem 4.2.3, $\sum_{i=1}^{k}{}^{mo} \mathcal{E}(P_i) \leq_{\mathcal{L}(\mathcal{K}_\mathcal{C})} \{F(X(\vec{u}))\} = \{\vec{u}\}$, which means that $\exists\, \vec{v} \in \sum_{i=1}^{k}{}^{mo} \mathcal{E}(P_i)$ such that $\vec{v} \leq_{\mathcal{K}_\mathcal{C}} \vec{u}$. Since $\vec{u}$ was an arbitrary element of $\mathcal{E}(P)$, it holds that $\sum_{i=1}^{k}{}^{mo} \mathcal{E}(P_i) \leq_{\mathcal{L}(\mathcal{K}_\mathcal{C})} \mathcal{E}(P)$.

4.4 Multiobjective weighted CSP

In this section we summarize the main definitions and results of the Chapter instantiated to the case of additive objective functions. Now, P_{mo} is a multi-objective optimization problem with p additive objective functions $F_j = \sum_{f \in \mathcal{F}_j} f$. The objective function of P_{mo} is $F = (F_1, \ldots, F_p)$.

A cost vector $\vec{v} = (v_1, \ldots, v_p)$ is a vector of p components where each $v_j \in \mathbb{N}^\infty$. Let $\vec{A}$ be the set of all possible cost vectors. Let $\vec{v}, \vec{u} \in \vec{A}$ be two distinct cost vectors. $\vec{v}$ *dominates* $\vec{u}$ (noted $\vec{v} < \vec{u}$), if $\overrightarrow{\min}\{\vec{v}, \vec{u}\} = \vec{v}$, where $\overrightarrow{\min}$ is the pointwise minimum of their components. The sum of cost vectors, noted $\overrightarrow{+}$, is defined as the pointwise sum of its components (i.e., $\vec{v} \overrightarrow{+} \vec{u} = (v_1 + u_1, \ldots, v_p + u_p)$).

Let S be a set of cost vectors. Its *set of non-dominated elements* is

$$\|S\| = \{\vec{v} \in S \mid \underset{\vec{u} \in S}{\forall}\ \vec{u} \not< \vec{v}\}$$

A set of non-dominated elements S (i.e., $S = \|S\|$) is called *frontier*.

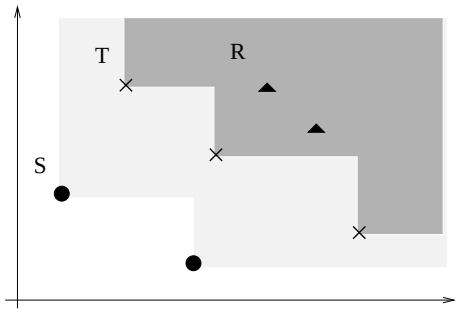

Figure 4.2: Domination and transitivity among frontiers.

Definition 4.4.1 *The valuation structure* $\mathcal{K}_{mo} = (A^{mo}, +^{mo}, \min^{mo})$ *is,*

- $A^{mo} = \{S \subseteq \vec{A} \mid S = \|S\|\}$
- $S +^{mo} T = \|\{\vec{v} \overrightarrow{+} \vec{u} \mid \vec{v} \in S, \vec{u} \in T\}\|$
- $\min^{mo}\{S, T\} = \|S \cup T\|$

Note that elements of A^{mo} are, by definition, non-dominated sets of cost vectors (i.e., frontiers).

The partial order over frontiers is,

Definition 4.4.2 $\forall\, S, T \in A^{mo}$ $S \leq_{\mathcal{K}_{mo}} T$ *iff* $\min^{mo}\{S, T\} = S$.

In words, frontier S is *better* than or *dominates* frontier T iff for all cost vectors $\vec{u} \in T$ there exists a cost vector $\vec{v} \in S$ such that $\vec{v}$ dominates $\vec{u}$. This notion of *preference* on frontiers will be crucial in multi-objective search (Chapters 5 and 6).

Example 4.4.1 *Consider bi-dimensional cost vectors. Figure 4.2 depicts three sets of vectors* S *as dots,* T *as crosses and* R *as triangles. All sets are frontiers since no point is dominated by any other of the same set. Frontier* S *dominates any frontier which vectors are in the light gray area. Therefore,* $S <_{\mathcal{K}_{mo}} T$ *(resp.* $T <_{\mathcal{K}_{mo}} R$*) because* $S \neq T$ *(resp.* $T \neq R$*) and the area*

dominated by S contains all the elements of T (resp. T contains all the elements of R).

Since $\leq_{\mathcal{K}_{mo}}$ is a partial order, it is transitive (i.e., $\forall\, S, T, R,\ S \leq_{mo} T \leq_{mo} R \Rightarrow S \leq_{mo} R$). The intuition (in a bi-objective problem) is as follows. Consider the 2D space and the frontier S in Figure 4.2. As shown in the previous example, frontier S dominates any frontier which vectors are in the light gray area. Let T be such a frontier. Frontier T dominates any frontier containing vectors in the dark gray area (e.g., frontier R). Since the dark gray area is contained in the light gray area and S dominates any frontier in the light gray area, it is clear that S also dominates any frontier in the dark gray area.

Definition 4.4.3 *A* frontier function *is a function $f : l(var(f)) \rightarrow A^{mo}$.*

As mentioned in Section 2.1, we extend the $+^{mo}$ and $\min^{mo}$ operators from valuations in A^{mo} to functions over A^{mo}.

Example 4.4.2 *Consider the biobjective frontier functions f and g in Figure 4.3 under domains $\{a, b\}$. The $\top$ value is $\{(\infty, \infty)\}$. The combination $f +^{mo} g$ is a biobjective function $(f +^{mo} g)(x_1, x_2, x_3)$ (see Figure 4.3). The elimination of variable x_3 from $f \otimes g$ is a biobjective function $\min_{x_3}^{mo}(f +^{mo} g)(x_1, x_2)$ (see Figure 4.3). Note that in $\min_{x_3}^{mo}(f +^{mo} g)(a, a)$, the cost vector $(4, 9)$ has been removed because it is dominated by the cost vector $(4, 4)$.*

Definition 4.4.4 *A* multiobjective weighted CSP *(MO-WCSP) is an optimization task $(\mathcal{X}, \mathcal{D}, A^{mo}, \mathcal{F}^{mo}, +^{mo}, \min^{mo})$ over c-semiring $\mathcal{K}_{mo} = (A^{mo}, +^{mo}, \min^{mo})$.*

The task of computing the efficient frontier $\mathcal{E}$ of problem P_{mo} can be expressed as a MO-WCSP problem $(\mathcal{X}, \mathcal{D}, A^{mo}, \mathcal{F}^{mo}, +^{mo}, \min^{mo})$ where $\mathcal{F}$ is a set of frontier functions defined as follows. Each function $f : l(var(f)) \rightarrow \mathbb{N}^{\infty}$ from P_{mo} belonging to the j^{th} criterion of P_{mo} is transformed into a new frontier function $f' : l(var(f)) \rightarrow A^{mo}$ defined as $f'(t) = \{(0, \ldots, 0, f(t), 0, \ldots, 0)\}$, where $f(t)$ is the j^{th} component of the singleton vector.

f:

x_1	x_2	
a	a	$\{(3,2),(2,8)\}$
a	b	$\{(4,10)\}$
b	a	$\top$
b	b	$\top$

g:

x_2	x_3	
a	a	$\{(1,2)\}$
a	b	$\{(2,1)\}$
b	a	$\{(6,2),(11,1)\}$
b	b	$\top$

$f +^{mo} g$:

x_1	x_2	x_3	
a	a	a	$\{(4,4),(3,10)\}$
a	a	b	$\{(5,3),(4,9)\}$
a	b	a	$\{(10,12),(15,11)\}$
a	b	b	$\top$
b	a	a	$\top$
b	a	b	$\top$
b	b	a	$\top$
b	b	b	$\top$

$\min_{x_3}^{mo}(f +^{mo} g)$:

x_1	x_2	
a	a	$\{(4,4),(3,10),(5,3)\}$
a	b	$\{(10,12),(15,11)\}$
b	a	$\top$
b	b	$\top$

Figure 4.3: Combination and marginalization over biobjective functions. $\top = \{(\infty, \infty)\}$.

Example 4.4.3 *Consider the bi-objective optimization problem P_{mo} of Example 4.3.1. Recall that the set of variables is $\mathcal{X} = \{x_1, x_2\}$ with domains $D_1 = D_2 = \{0, 1\}$. The set of functions for the first and second objective functions is $\mathcal{F}_1 = \{f_1(x_1) = x_1, f_2(x_2) = x_2\}$ and $\mathcal{F}_2 = \{h_1(x_1) = 1 - x_1, h_2(x_2) = x_2\}$, respectively.*

This problem can be modeled as a MO-WCSP problem

$$P = (\{x_1, x_2\}, \{0, 1\}, A^{mo}, \mathcal{F}^{mo}, +^{mo}, \min^{mo})$$

The set of frontier functions is $\mathcal{F}^{mo} = \{f_1'(x_1), f_2'(x_2), h_1'(x_1), h_2(x_2)\}$ where,

$$f_1'(x_1) = \{(x_1, 0)\} \qquad f_2'(x_2) = \{(x_2, 0)\}$$
$$h_1'(x_1) = \{(0, 1 - x_1)\} \qquad h_2'(x_2) = \{(0, x_2)\}$$

The objective function of P is $F^{mo}(\mathcal{X}) = \sum_{f \in \mathcal{F}^{mo}}^{mo} f$. $F^{mo}(\mathcal{X})$ can be expressed extensionally as the following table,

x_1	x_2	
0	0	$\{(0,1)\}$
0	1	$\{(1,2)\}$
1	0	$\{(1,0)\}$
1	1	$\{(2,1)\}$

The reasoning task defined by P is,

$$\min_{\mathcal{X}}{}^{mo}\{F^{mo}(\mathcal{X})\}$$

The result of computing the previous expression is $\{(0,1),(1,0)\}$, which is the efficient frontier of P_{mo}. Note that $(1,2)$ and $(2,1)$ are not efficient, because they are dominated by either $(0,1)$ or $(1,0)$.

In the following chapters we focus on the resolution of MO-WCSP problems. For readability reasons, we will write $+$, *min* and $<$ instead of $+^{mo}$, min^{mo} and $<_{mo}$, respectively, when it is clear by the context.

4.5 Conclusions

In this Chapter we have described multi-objective tasks within the semiring CSP framework. This means that multi-objective tasks can be axiomatically described in terms of a partially ordered c-semiring. For the first time, we have proposed a SCSP instance over a new c-semiring able to compute the efficient frontier of a multi-objective problem. Moreover, we show that previous attempts described in the literature were not completely satisfactory since they only characterized the greatest lower bound of the efficient frontier.

We do not make any assumption about the nature of the objectives in the problem. The only requirement is that each objective function can be independently expressed as an instance of the SCSP framework. Therefore, our formalization is valid for any multi-objective problem satisfying this assumption.

Chapter 5

Branch and Bound

Branch and Bound (BB) is a well-known search schema typically used to solve mono-objective optimization problems. The purpose of this chapter is to extend depth-first BB from mono-objective to multi-objective optimization. Essentially, we extend the three main elements of BB: the *lower bound*, the *upper bound* and the *pruning condition*. Interestingly, the extension of these three notions follow naturally the results of Chapter 4: *bounds* are valuations of the $\mathcal{L}(\mathcal{K}_\mathcal{C})$ c-semiring and the *pruning condition* is naturally expressed in terms of its induced partial order $\leq_{mo}$. Apparently, the resulting algorithm is identical to the mono-objective case, although the low level details are completely different because frontiers rather than scalar values are used, along with the operators $+^{mo}$ and $\min^{mo}$ of multi-objective WCSP (MO-WCSP) problems which are much more complex than their mono-objective counterpart. It is the merit of the graphical model framework to unify in such an elegant way mono-objective and multi-objective BB.

The structure of the chapter is as follows. Section 5.1 recalls depth-first branch-and-bound search for WCSP problems along with some classic lower bounds from the literature. Then, Section 5.2 formally defines what a multi-objective bound is and presents the extension of depth-first branch-and-bound to solve MO-WCSP problems. Moreover, it presents some direct extensions of the classic mono-objective lower bounds to the multi-objective

context. Section 5.3 shows some experimental results. Section 5.4 discusses related work and points out its differences with respect to our extension. Finally, Section 5.5 gives some conclusions.

5.1 (Mono-objective) Branch-and-Bound

Branch-and-Bound Search (BB) [53] is a general search algorithm for combinatorial optimization. Given an optimization problem P, BB systematically enumerates all its total assignments, discarding large subsets of fruitless complete assignments by using upper and lower bounds of the criteria being optimized. The set of total assignments is the search space. It can be represented as a tree: Given an arbitrary node, its children represent the assignment of one new variable to each one of its domain values. Then, each node represents an assignment where only the variables included in the path from the root to this node have been assigned. Each node roots a subtree where the variables in the path from the node to the leaves remains unassigned. This subtree represents a subproblem of P resulting from instantiating the set of functions in P to t (i.e., $P(t)$), where t is the assignment associated with that node. When the node is a leaf, t is complete and $P(t)$ is composed by a set of totally instantiated functions. The combination of these functions (i.e., sum of costs) is the cost of t.

When BB solves problem P, it traverses the search tree in a specific order (e.g., depth-first, breath-first). During the traversal, the branch-and-bound schema stores the cost of the best complete assignment found so far. This cost is an *upper bound* (`ub`) of the problem optimum ($\texttt{opt}(P)$) (i.e., $\texttt{opt}(P) \leq \texttt{ub}$). Consider an arbitrary search node and let t be its associated assignment. The subproblem rooted at this node (i.e., $P(t)$) does not lead to a better solution if its optimum ($\texttt{opt}(P(t))$) cannot improve the best cost found so far (i.e., $\texttt{ub} \leq \texttt{opt}(P(t))$). In order to foresee this situation without solving $P(t)$, BB computes an underestimation or lower bound (`lb`) of the optimum of $P(t)$ (i.e., $\texttt{lb} \leq \texttt{opt}(P(t))$). If the lower bound is higher than

procedure DF-BB($(\mathcal{X}, \mathcal{D}, \mathcal{F})$, ub)
1. **if** $\mathcal{X} = \emptyset$ **then**
2. ub := $\sum_{f \in \mathcal{F}} f()$; /* Note that $\forall f \in \mathcal{F}, var(f) = \emptyset$ */
3. **else**
4. $x_i := Select(\mathcal{X})$;
5. **for each** $a \in D_i$ **do**
6. $\mathcal{F}' := \{f(x_i = a) \mid f \in \mathcal{F}\}$; $\mathcal{X}' := \mathcal{X} - \{x_i\}$; $\mathcal{D}' := \mathcal{D} - \{D_i\}$;
7. $P' := (\mathcal{X}', \mathcal{D}', \mathcal{F}')$; /* P' is the new current subproblem */
8. lb := LB(P');
9. **if** lb $<$ ub **then** DF-BB (P', ub);
10. **endfor**
11. **endif**

endprocedure

Figure 5.1: Depth-first Branch-and-Bound Algorithm. The input of the algorithm is a WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ and an upper bound ub. The algorithm returns the optimal cost of P in ub.

or equal to the upper bound (i.e., ub $\leq$ lb), it is clear that exploring $P(t)$ is useless. Formally,

$$\texttt{ub} \leq \texttt{lb} \wedge \texttt{lb} \leq \texttt{opt}(P(t)) \Rightarrow \texttt{ub} \leq \texttt{opt}(P(t))$$

When that is the case, BB prunes the current subtree (i.e., it discards the current search branch) and backtracks to a previous node. Note that the condition to continue the search is ub $\not\leq$ lb, which is equivalent to lb $<$ ub due to the total order $<$ on naturals. When the whole search tree is traversed, ub is the optimal cost of the problem P.

Figure 5.1 shows a recursive description of *depth-first branch-and-bound* (DF-BB) algorithm for WCSP problems. The input of the algorithm is a WCSP instance $(\mathcal{X}, \mathcal{D}, \mathcal{F})$, and an upper bound ub. In the initial call,

$(\mathcal{X}, \mathcal{D}, \mathcal{F})$ is the original problem P and `ub` is set to a known upper bound for its optimal solution (∞ if a better bound is not known). In each recursive call, the algorithm tries to assign one new variable in order to move down in the search tree. Then, in an arbitrary call, $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ is a subproblem $P(t)$, where t is the current assignment, and `ub` is the best cost found so far. When $\mathcal{X}$ is empty (lines 1-3), all the variables has been assigned, so the algorithm has reached a leaf node. Then, the cost of the current total assignment (stored in $\mathcal{F}$ as a set of totally assigned functions) is the best one found so far. Therefore, `ub` is updated (line 2). When $\mathcal{X}$ is not empty (lines 4-11), there exist some unassigned variables, so the algorithm is in an internal search node. Then, DF-BB selects an unassigned variable (line 4) and sequentially attempts the assignment of its domain values (line 5). Each assignment $x_i = a$ leads to a new subproblem P' (line 7) of $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ resulting from conditioning the cost functions in $\mathcal{F}$ to the current assignment and removing variable x_i and its domain D_i from $\mathcal{X}$ and $\mathcal{D}$, respectively (line 6). The algorithm computes a lower bound `lb` of P' using a bounding evaluation function LB (line 8). If the `ub` is better or equal than the `lb`, the algorithm prunes the current line of search since it does not lead to a better solution. Otherwise, the algorithm proceeds recursively (line 9). When the whole search tree is traversed, clearly $\texttt{ub} = \texttt{opt}(P)$.

The performance of the search algorithm can be increased by improving its pruning capabilities. This reduction greatly depends on the *bounding evaluation function* LB. Tighter bounds allows the algorithm to prune earlier in the search tree, thus reducing the number of visited nodes. In general, more computational effort results in better bounds. However, since the algorithm computes a lower bound in every visited node, there is a trade-off between the computational overhead and the pruning capability.

5.1.1 Basic mono-objective lower bounds

Many lower bounds have been proposed in the mono-objective optimization context. In this Section, we outline some basic approaches proposed for WC-

SPs, which are the basis for more elaborated ones. Chapter 6 and Chapter 8 will describe two more sophisticated lower bounds.

Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ be the original WCSP problem to be solved. Consider an arbitrary node being visited by DF-BB and let t be its associated partial assignment. At this point, functions in $\mathcal{F}$ has been conditioned to t. Namely, $P(t)$ is the current WCSP subproblem to be solved. The simplest lower bound $\mathtt{lb}_s$ considers the sum of costs of every totally instantiated function,

$$\mathtt{lb}_s = \sum_{\substack{f \in P(t) \\ |var(f)|=0}} f()$$

This lower bound can be improved by considering necessary costs from extending the current assignment to unassigned variables. The most usual approach is to consider the minimum contribution from the extensions to one new variable [53]. Let $x_k \notin var(t)$ be an unassigned variable and $a \in D_k$ one of its domain values. The *inconsistency count* associated with domain value a of variable x_k (noted $\mathtt{ic}_{ka}$) is the cost of extending the current assignment t to one new variable x_k taking domain value $a \in D_k$. Namely,

$$\mathtt{ic}_{ka} = \sum_{\substack{f \in P(t) \\ var(f)=\{x_k\}}} f(x_k = a)$$

Then, the inconsistency count associated to variable x_k (noted $\mathtt{ic}_k$), is the necessary cost to extend the current assignment t to x_k, no matter what domain value is assigned to variable x_k. Namely,

$$\mathtt{ic}_k = \min_{a \in D_k} \{\mathtt{ic}_{ka}\}$$

The sum of inconsistency counts, $\sum_{x_k \notin var(t)} \mathtt{ic}_k$ is a lower bound of the cost that will necessary have any extension of t to a complete assignment. Therefore, it can be added to $\mathtt{lb}_s$ in order to obtain a better lower bound at the current node. Thus,

$$\mathtt{lb}_{ic} = \mathtt{lb}_s + \sum_{x_k \notin var(t)} \mathtt{ic}_k$$

is the lower bound computed when considering the inconsistency counts.

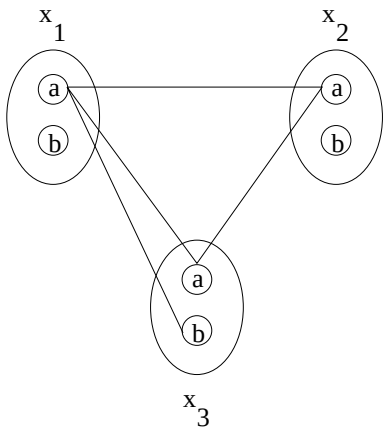

Figure 5.2: WCSP instance.

Example 5.1.1 *Consider the WCSP instance of Figure 5.2. It has three variables* $\{x_1, x_2, x_3\}$ *and two domain values per domain* $D_i = \{a, b\}$*. There are three binary cost functions:* $f_{x_1x_2}(x_1, x_2)$*,* $f_{x_1x_3}(x_1, x_3)$ *and* $f_{x_2x_3}(x_2, x_3)$*. All unary costs are* 0*. Binary costs are* 1 *when there exists an edge connecting the corresponding pair of values. Otherwise, the cost is* 0*. Let* $t = (x_1 = a, x_2 = a)$ *be the current partial assignment. At this point in search, the simplest lower bound is* $\mathtt{lb}_s = 1$ *because there is only one totally assigned function (i.e.,* $f_{x_1x_2}$*) and its valuation in* t *is* $f_{x_1x_2}(t) = 1$*. The only unassigned variable is* x_3*. Its inconsistency counts are as follows.* $ic_{3a} = 2$ *because there are two functions with scope* $\{x_3\}$ *after they are partially assigned with* t *(i.e.,* $f_{x_1x_3}(t)$ *and* $f_{x_2x_3}(t)$*) and* $f_{x_1x_3}(t \cdot x_3 = a) + f_{x_2x_3}(t \cdot x_3 = a) = 2$*. Similarly,* $ic_{3b} = 1$ *because* $f_{x_1x_3}(t \cdot x_3 = b) + f_{x_2x_3}(t \cdot x_3 = b) = 1$*. Any extension of the current partial assignment to variable* x_3 *will have, at least, a cost of* 1 *(i.e.,* $\mathtt{ic}_3 = \min\{ic_{3a}, \mathtt{ic}_{3b}\}$*). Therefore, the lower bound considering inconsistency counts is* $\mathtt{lb}_{ic} = 1 + 1 = 2$*.*

5.2 Multi-objective Branch and Bound

Multi-objective problems can also be solved with a branch-and-bound schema. The idea of the multi-objective approach is the same as for the mono-objective case. Given a multi-objective optimization problem P, the algorithm enumerates all possible complete assignments, which can be represented as a tree, and tries to prune branches that cannot lead to a new optimal solution. Each node represents an assignment t and roots a multi-objective optimization problem $P(t)$. However, there exist some differences, that we discuss in the following.

During search, mono-objective BB maintains the cost `ub` of the best solution found so far. Since multi-objective optimization is characterized by a set of optimal solutions, the algorithm must store a set of cost vectors `ubf`. Each vector in `ubf` is the valuation of one solution which is not dominated by any other found so far. Therefore, it is a candidate for being part of the efficient frontier of P ($\mathcal{E}(P)$). When a new solution is found, in mono-objective BB the value of `ub` is updated because the old value cannot be the optimum. In multi-objective optimization the value of `ubf` is also updated by adding the cost vector $\vec{v}$ of the new solution to `ubf` and by removing all those that are dominated because they cannot be part of the efficient frontier $\mathcal{E}(P)$. According with our notation, the updating of `ubf` can be expressed as,

$$\texttt{ubf} = \min^{mo}\{\texttt{ubf}, \vec{v}\}$$

Note that each cost vector $\vec{w}$ in `ubf` comes from a complete assignment. Moreover, during search $\vec{w}$ will either be removed because a better solution has been found or it will remain because it is part of $\mathcal{E}(P)$. Then, by construction of `ubf`, it is easy to see that the following condition is an invariant,

$$\underset{\vec{w}\in\texttt{ubf}}{\forall} \underset{\vec{v}\in\mathcal{E}(P)}{\exists} \vec{v} \leq \vec{w}$$

In words, all vectors in `ubf` are dominated by or equal to at least one vector in the efficient frontier of the problem $\mathcal{E}(P)$. According to the partial order

on frontiers (see Definition 4.4.2), this is equivalent to

$$\mathcal{E}(P) \leq_{mo} \texttt{ubf}$$

The clear parallelism between the role of `ub` in BB and the role of `ubf` in multi-objective BB leads us to call `ubf` the *upper bound frontier*.

Definition 5.2.1 *Given a MO-WCSP problem P, we say that a frontier $\mathtt{S}$ is a (*valid*) upper bound frontier iff $\mathcal{E}(P) \leq_{mo} \mathtt{S}$, where $\mathcal{E}(P)$ is the efficient frontier of P.*

Consider an arbitrary search node and let t be its associated assignment. In mono-objective optimization, the current subproblem $P(t)$ can be safely pruned (i.e., discarded) when it cannot lead to a better solution. That is the case when the optimal solution of $P(t)$ is greater than or equal to the best found so far (i.e., $\texttt{ub} \leq opt(P(t))$). In multi-objective optimization, $P(t)$ can be discarded when it *cannot lead to* any efficient solution. That is the case when all efficient solutions of $P(t)$ are dominated by or equal to any other found so far. In other words, when each cost vector in $\mathcal{E}(P(t))$ is dominated by or equal to at least one vector in `ubf`. Formally,

$$\texttt{ubf} \leq_{mo} \mathcal{E}(P(t))$$

In order to foresee this situation, mono-objective BB computes a lower bound `lb` of the optimum of $P(t)$. In multi-objective optimization, the algorithm must compute an underestimation `lbf` of the efficient frontier of $P(t)$ (i.e., $\mathcal{E}(P(t))$). The necessary condition is that each cost vector in $\mathcal{E}(P(t))$ must be dominated by at least one cost vector in `lbf`. This property can be formally written as,

$$\texttt{lbf} \leq_{mo} \mathcal{E}(P(t))$$

This condition can be seen as very demanding. We could think of a relaxed condition where it is enough for `lbf` to contain at least one cost vector that dominates one cost vector in $\mathcal{E}(P(t))$. However, as we will show, this alternative definition will not lead to a sufficient condition for pruning. Again,

the clear parallelism between the role of `lb` in BB and the role of `lbf` in multi-objective BB leads us to call `lbf` the *lower bound frontier*.

Definition 5.2.2 *Given a MO-WCSP problem P, we say that a frontier $\mathcal{S}$ is a* valid *lower bound frontier iff $\mathcal{S} \leq_{mo} \mathcal{E}(P)$, where $\mathcal{E}(P)$ is the efficient frontier of P.*

According with this definition, $\texttt{lbf}(P(t))$ can be computed by any bounding evaluation function such that $\texttt{lbf}(P(t)) \leq_{mo} \mathcal{E}(P(t))$.

Since $\texttt{lbf} \leq_{mo} \mathcal{E}(P(t))$ is true by definition, if all cost vectors in `lbf` are dominated by at least one vector in `ubf` (i.e., $\texttt{ubf} \leq_{mo} \texttt{lbf}$), it is easy to see that all cost vectors in $\mathcal{E}(P(t))$ will also be dominated by some vector in `ubf` (i.e., $\texttt{ubf} \leq_{mo} \mathcal{E}(P(t))$). Formally,

$$\texttt{ubf} \leq_{mo} \texttt{lbf} \wedge \texttt{lbf} \leq_{mo} \mathcal{E}(P(t)) \Rightarrow \texttt{ubf} \leq_{mo} \mathcal{E}(P(t))$$

When that is the case, the algorithm prunes the subproblem $P(t)$ and backtracks to a previous node. It is worth noting that the previous statement holds bacause the partial order $\leq_{mo}$ is transitive (as proved in Theorem 4.2.1).

It is also important to remark that $\texttt{ubf} \leq_{mo} \texttt{lbf}$ is a sufficient condition to prune because $\texttt{lbf} \leq_{mo} \mathcal{E}(P(t))$ is true by definition. Any other weaker definition of `lbf` will be insufficient for the algorithm to decide whether to backtrack or not, as we demonstrate in the following. Consider a frontier `lbf` and let $\mathcal{E}(P(t))$ be divided into two sets: S is the subset of vectors dominated by some vector in `lbf` (i.e., $\texttt{lbf} \leq_{mo} S$), and T is the subset of vectors not dominated by any vector in `lbf` (i.e., $\texttt{lbf} \not\leq_{mo} T$). The search algorithm will prune subproblem $P(t)$ when $\texttt{ubf} \leq_{mo} S \wedge \texttt{ubf} \leq_{mo} T$. It is easy to see that,

$$\texttt{ubf} \leq_{mo} \texttt{lbf} \wedge \texttt{lbf} \leq_{mo} S \Rightarrow \texttt{ubf} \leq_{mo} S$$

However, from

$$\texttt{ubf} \leq_{mo} \texttt{lbf} \wedge \texttt{lbf} \not\leq_{mo} T$$

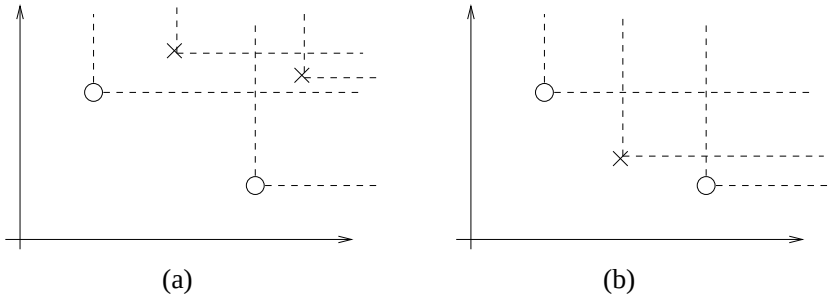

Figure 5.3: (a) Backtrack and (b) continue condition in multi-objective branch-and-bound algorithm for a bi-objective optimization problem. The current `ubf` and `lbf` are depicted as dots and crosses, respectively.

we cannot conclude neither that $\texttt{ubf} \not\leq_{mo} T$ nor $\texttt{ubf} \leq_{mo} T$. As a consequence, we cannot conclude whether $\texttt{ubf} \leq_{mo} \mathcal{E}(P(t))$ holds or not unless $T = \emptyset$. When $T = \emptyset$, then $\mathcal{E}(P(t)) = S$ and $\texttt{lbf} \leq_{mo} \mathcal{E}(P(t))$, which is exactly the condition we impose for a frontier to be a (valid) lower bound frontier.

Example 5.2.1 *Consider a bi-objective optimization problem being solved by multi-objective branch-and-bound. Figure 5.3 (a) shows the* `ubf` *(depicted as dots) and* `lbf` *(depicted as crosses) at the current search node. Dotted lines show the domination area of each cost vector. Note that* `ubf` *dominates* `lbf`*. By definition, the efficient frontier $\mathcal{E}'$ of the subproblem rooted at the current node will be dominated by* `lbf`*. As a consequence,* `ubf` *will also dominate $\mathcal{E}'$, which means that $\mathcal{E}'$ does not contain any possible candidate to be part of the efficient frontier of the original problem. Then, the algorithm can safely backtrack because it will not discard any potential efficient solution.*

It is also important to note that the condition to continue the search is $\texttt{ubf} \not\leq_{mo} \texttt{lbf}$, which is not equivalent to $\texttt{lbf} \leq_{mo} \texttt{ubf}$ because $\leq_{mo}$ is a partial order.

Example 5.2.2 *Consider the bi-objective optimization problem of the previ-*

procedure MO-BB$((\mathcal{X}, \mathcal{D}, \mathcal{F}), \texttt{ubf})$
1. **if** $\mathcal{X} = \emptyset$ **then**
2. $\texttt{ubf} := \min^{mo}\{\texttt{ubf}, \sum_{f \in \mathcal{F}}^{mo} f()\}$;
3. **else**
4. $x_i := Select(\mathcal{X})$;
5. **for each** $a \in D_i$ **do**
6. $\mathcal{F}' := \{f(x_i = a) \mid f \in \mathcal{F}\}$; $\mathcal{X}' := \mathcal{X} - \{x_i\}$; $\mathcal{D}' := \mathcal{D} - \{D_i\}$;
7. $P' := (\mathcal{X}', \mathcal{D}', \mathcal{F}')$;
8. $\texttt{lbf} := \text{LBF}(P')$;
9. **if** $\texttt{ubf} \not\leq_{mo} \texttt{lbf}$ **then** MO-BB $(P', \texttt{ubf})$;
10. **endfor**
11. **endif**

endprocedure

Figure 5.4: Multi-objective Depth-First Branch-and-Bound Algorithm. The input of the algorithm is a MO-WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ and an upper bound frontier ubf. The algorithm returns the efficient frontier of P in ubf.

ous example. Figure 5.3 (b) shows the $\texttt{ubf}$ *and* $\texttt{lbf}$ *at another search node. Consider that* $\texttt{lbf} = \{\vec{v}\}$ *is exactly the efficient frontier of the subproblem rooted at that node. In this situation,* $\vec{v}$ *is a new potential element of the efficient frontier of the original problem because it is not dominated by any cost vector in* $\texttt{ubf}$*. As a consequence, the algorithm should continue the search. However, if the continue condition was* $\texttt{lbf} \leq_{mo} \texttt{ubf}$*, the algorithm would backtrack, missing this potential efficient solution. The reason is that the previous condition does not capture the fact that a cost vector in* $\texttt{lbf}$ *that is not dominated by any other in* $\texttt{ubf}$ *represents a potential new candidate to be part of the efficient frontier.*

MO-BB (Figure 5.4) is a recursive description of *depth-first multi-objective branch-and-bound* algorithm for MO-WCSP problems. The input of the al-

gorithm is a MO-WCSP instance $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ and an upper bound frontier `ubf`. In the initial call, $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ is the original problem P and `ubf` is set to a known upper bound frontier ($\{(\infty, \ldots, \infty)\}$ if a better bound in not known). In an arbitrary call, $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ is a subproblem $P(t)$, where t is the current assignment, and `ubf` is the set of non-dominated cost vectors found so far. In each recursive call, the algorithm tries to assign one new variable. First of all, if no variable remains, all the functions in $\mathcal{F}$ are totally assigned and $\sum_{f \in \mathcal{F}}^{mo} f()$ is a frontier with one vector. In that case, `ubf` is updated with the cost vector of the current assignment using the $\min^{mo}$ operator (line 2). If $\mathcal{X}$ is not empty, an arbitrary unassigned variable x_i is selected (line 4). Then, the algorithm sequentially attempts the assignment of its domain values (line 5). Each assignment leads to a new subproblem P' (line 8) resulting from conditioning the functions in $\mathcal{F}$ to the current assignment and removing x_i and D_i from $\mathcal{X}$ and $\mathcal{D}$, respectively (lines 6-7). After each assignment, the algorithm computes a lower bound frontier `lbf` of P' with the bounding evaluation function LBF (line 9). If the pruning condition does not hold (line 10), the search procedure proceeds by making a recursive call (line 11). Otherwise, the algorithm backtracks since the current search branch does not lead to any new candidate to be part of the efficient frontier. When the whole search tree is traversed, clearly `ubf` $= \mathcal{E}(P)$.

Theorem 5.2.1 *During the execution of the algorithm* `ubf` *is a valid upper bound frontier.*

Proof The theorem is an obvious consequence of Theorem 4.3.4. By construction, `ubf` is a set of non-dominated elements coming from a subset of complete assignments while $\mathcal{E}(P)$ is a set of non-dominated elements coming from all complete assignments.

It is worth noting that the structure of DF-BB (Figure 5.1) and MO-BB (Figure 5.4) is identical. This observation is hardly a surprise because the graphical model framework provides a unifying view of algorithms developed

for graphical model instances. However, it is also important to note that there are significant differences encapsulated in the particular instantiation of the operators used to combine, choose and compare valuations.

Property 5.2.1 *When considering a WCSP problem (i.e., $p = 1$), the algorithm MO-BB is equivalent to BB.*

Proof When $p = 1$, cost vectors have one component. Then, the partial order $\leq_{mo}$ reduces to the usual total order $<$ among naturals. Therefore, $\min^{mo}$ is equivalent to min, $\texttt{ubf} \not\leq_{mo} \texttt{lbf}$ is equivalent to $ub > lb$, and $\texttt{ubf} \leq_{mo} \mathcal{E}(P(t))$ is equivalent to $\texttt{ub} \leq \texttt{opt}(P(t))$.

The performance of the search algorithm depends on the quality of the *upper* and *lower bound frontiers*, because tight bounds prune at early stages of the search. The simplest `ubf` is $\{(\infty, \ldots, \infty)\}$. Better approximations can be obtained, for example, with multi-objective local search heuristics [41]. Regarding lower bound frontiers, we propose in the next Section some basic `lbf` resulting from adapting the classic mono-objective optimization bounds described in Section 5.1.1.

5.2.1 Basic multi-objective lower bounds

Classic mono-objective optimization bounds can be easily extended to the multi-objective context. They also combine the contributions of assigned and unassigned variables. It is important to note that in this context those contributions are frontiers and the operators are the corresponding defined in the multi-objective context. As their mono-objective counterparts, these bounds are the basis for more elaborated ones (see Chapters 6 and 8).

Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ be the MO-WCSP problem to be solved. Consider an arbitrary search node and let t be the current partial assignment. $P(t)$ is the current MO-WCSP subproblem to be solved. The simplest `lbf` is a singleton frontier resulting from the sum of cost vectors from totally instantiated

frontier functions,

$$\texttt{lbf}_s = \sum_{\substack{f \in P(t) \\ |var(f)|=0}}^{mo} f()$$

Essentially, $\texttt{lbf}_s$ is a frontier with one cost vector.

The idea of inconsistency counts described for mono-objective optimization can be also extended to multi-objective optimization. In this case,

$$\texttt{icf}_{ka} = \sum_{\substack{f \in P(t) \\ var(f)=\{x_k\}}}^{mo} f(x_k = a)$$

is the multi-objective cost (i.e., cost vector) of assigning one new variable x_k to its domain value $a \in D_k$. Note that $\texttt{icf}_{ka}$ is a frontier with one cost vector. Then, the best costs that can be obtained if x_k is assigned to any of its domain values is

$$\texttt{icf}_k = \min_{a \in D_k}^{mo} \{\texttt{icf}_{ka}\}$$

It is worth noting that, in this case, $\texttt{icf}_k$ is a frontier with (possibly) many cost vectors. Each incomparable alternative comes from a different domain value $a \in D_k$.

Likewise, the combination of the $\texttt{icf}_k$ for all unassigned variable x_k is the set of best incomparable alternatives that can be obtained when extending the current assignment to a total one, no matter what domain values are assigned to unassigned variables. Since the subset of functions taken into account by this frontier (functions with just one unassigned variable) is disjunct with respect the subset of functions taken into account by $\texttt{lbf}_s$ (completely assigned functions), they can be added in order to obtain a tighter lower bound frontier at the current node. Then,

$$\texttt{lbf}_{ic} = \texttt{lbf}_s +^{mo} \sum_{x_k \notin var(t)}^{mo} \texttt{icf}_k$$

is a lower bound frontier of $P(t)$.

Theorem 5.2.2 *The frontier $\texttt{lbf}_{ic}$ is a valid lower bound frontier.*

Proof Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ be the original MO-WCSP problem and t be the current partial assignment. Thus, $P(t)$ is the current subproblem solved by MO-BB. For clarity reasons, $P(t)$ will be called P'. Let $\mathcal{F}'$ and $\mathcal{X}'$ be the set of functions and the set of unassigned variables in P', respectively. Let $\mathcal{E}(P')$ be its efficient frontier. We have to prove that $\texttt{lbf}_{ic} \leq_{mo} \mathcal{E}(P')$.

The set of functions $\mathcal{F}'$ can be partitioned into three sets: completely assigned functions, unary functions, and functions with more than one variable in its scope. We call P'_1, P'_2 and P'_3 the induced problems of each subset, respectively. By Theorem 4.3.4,

$$\mathcal{E}(P'_1) +^{mo} \mathcal{E}(P'_2) +^{mo} \mathcal{E}(P'_3) \leq_{mo} \mathcal{E}(P') \tag{5.1}$$

It is easy to see that $\mathcal{E}(P'_1) = \texttt{lbf}_s$. The efficient frontier of P'_2 is,

$$\mathcal{E}(P'_2) = \min_{X \in l(\mathcal{X}')}^{mo} \{ (\sum_{\substack{f \in \mathcal{F}' \\ |var(f)|=1}}^{mo} f)(X) \}$$

Since it only contains unary functions, it can be rewritten as,

$$\mathcal{E}(P'_2) = \sum_{x_k \notin var(t)}^{mo} \min_{a \in D_k}^{mo} \{ \sum_{\substack{f \in \mathcal{F}' \\ var(f)=\{x_k\}}}^{mo} f(x_k = a) \} = \sum_{x_k \notin var(t)}^{mo} \texttt{icf}_k$$

Expression (5.1) can be rewritten as,

$$\texttt{lbf}_s +^{mo} \sum_{x_k \notin var(t)}^{mo} \texttt{icf}_k +^{mo} \mathcal{E}(P'_3) = \texttt{lbf}_{ic} +^{mo} \mathcal{E}(P'_3) \leq_{mo} \mathcal{E}(P') \tag{5.2}$$

Since $+^{mo}$ is intensive (see Theorem 4.2.4),

$$\texttt{lbf}_{ic} \leq_{mo} \texttt{lbf}_{ic} +^{mo} \mathcal{E}(P'_3) \tag{5.3}$$

By monotonicity of $\leq_{mo}$ (see Theorem 4.2.2) on expressions (5.3) and (5.2) ,

$$\texttt{lbf}_{ic} \leq_{mo} \mathcal{E}(P')$$

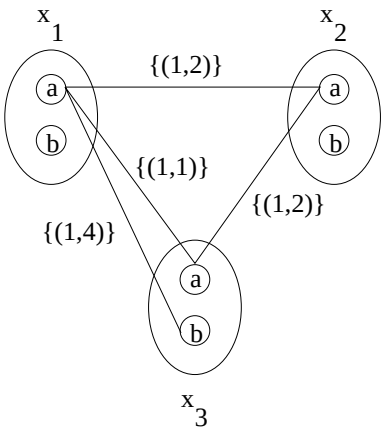

Figure 5.5: MO-WCSP instance.

Example 5.2.3 *Consider the MO-WCSP instance of Figure 5.5. It is based on the WCSP of Example 5.2 adding one new objective. The problem has three variables* $\{x_1, x_2, x_3\}$ *and two domain values per domain* $D_i = \{a, b\}$. *There are three binary frontier functions:* $f_{x_1x_2}(x_1, x_2)$, $f_{x_1x_3}(x_1, x_3)$ *and* $f_{x_2x_3}(x_2, x_3)$. *All unary valuations are* $\{(0, 0)\}$. *Binary valuations are depicted as frontier labeled edges connecting the corresponding pair of values. Only non-zero frontiers are shown. Let* $t = (x_1 = a, x_2 = a)$ *be the current partial assignment. At this point in search, the simplest lower bound frontier is* $\mathtt{lbf}_s = \{(1, 2)\}$ *because the only totally assigned function is* $f_{x_1x_2}$, *and its valuation in* t *is* $f_{x_1x_2}(a, a) = \{(1, 2)\}$. *The only unassigned variable is* x_3. *Its inconsistency counts are as follows.* $\mathtt{icf}_{3a} = \{(2, 3)\}$ *because* $f_{x_1x_3}(t \cdot x_3 = a) + f_{x_2x_3}(t \cdot x_3 = a) = \{(1, 1)\} + \{(1, 2)\}$. $\mathtt{icf}_{3b} = \{(1, 4)\}$ *because* $f_{x_1x_3}(t \cdot x_3 = b) + f_{x_2x_3}(t \cdot x_3 = b) = \{(1, 4)\}$. *Therefore,* $\mathtt{icf}_3 = \min^{mo}\{\mathtt{icf}_{3a}, \mathtt{icf}_{3b}\} = \{(1, 4), (2, 3)\}$ *(i.e.,). Then,* $\mathtt{lbf}_{ic} = \{(1, 2)\} +^{mo} \{(1, 4), (2, 3)\} = \{(2, 6), (3, 5)\}$.

5.3 Experimental Results

The goal of these experiments is to give a first insight on the suitability of MO-BB using a basic multi-objective lower bound to solve MO-WCSP problems. In particular, we assess the adequacy of MO-BB using the lower bound frontier $\texttt{lbf}_{\texttt{icf}}$, noted MO-$\text{BB}_{\texttt{icf}}$. It is worth noting that, although being multi-objective in nature (i.e., it is a frontier rather than a single vector of costs), $\texttt{lbf}_{\texttt{icf}}$ is very naïve. We use an implementation based on Toolbar[1].

We compare the performance of MO-$\text{BB}_{\texttt{icf}}$ versus the following two alternative multi-objective algorithms:

- ϵ-constraint. As described in Section 3.2.2, this algorithm transforms the multi-objective problem into a sequence of CSPs. In our experiments, we use Ilog Solver 6.1[2] as CSP solver. Recall that ϵ-constraint receives as a parameter a lower bound on each objective function when considered independently, noted (l_1, l_2). In our experiments, we run the algorithm with l_1 and l_2 being the optimal costs on each objective function. The time spent to compute them is not taken into account.

- MO-BB_{fdac}. MO-BB enforcing FDAC [88] independently in each objective function. This lower bound has been proved very efficient in mono-objective optimization. Note that the lower bound computed by FDAC is a singleton vector of costs. The components of the vector approximate the optimal cost of each objective function when considered independently. It is worth noting that, although being much more sophisticated than $\texttt{lbf}_{\texttt{icf}}$, FDAC is mono-objective in nature. We use an implementation based on Toolbar.

We test their performance in four different domains: *Max-SAT-ONE*, *biobjective minimum vertex cover*, *risk-conscious combinatorial auctions* and *scheduling of an EOS* benchmarks (for a detailed description of each benchmark see Appendices B.3, B.4, B.1, B.2, respectively).

[1] http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro
[2] http://www.ilog.fr

The time limit for all the experiments is 1800 seconds.

As we will see next, the main conclusion from these experiments is that it is reasonable to develop (more sophisticated) pure multi-objective optimization algorithms to deal with multi-objective optimization problems. The following Chapters are devoted to those new algorithms.

5.3.1 Max-SAT-ONE

Figure 5.6 shows the results for the Max-SAT-ONE instances. We only report the results for the instances solved within the time limit by at least one of the three studied algorithms. The first, second and third columns contain the name of the instance, the number of variables and the number of clauses, respectively. The fourth column indicates the size of the efficient frontier of each problem. The last three columns contain the cpu time in seconds required by the three algorithms. A "-" indicates that the algorithm does not terminate in 1800 seconds.

In general, MO-BB$_{\texttt{icf}}$ outperforms the other two approaches. Notably, it is the only algorithm able to solve *aim–50–6_0* instances within the time limit. The improvement of MO-BB$_{\texttt{icf}}$ with respect to ϵ-constraint is up to various orders of magnitude (see *aim* instances). Its performance with respect to MO-BB$_{fdac}$ is not so prominent. However, for some instances, as for example aim–50–2_0–yes1–4, aim–50–3_4–yes1–3 and aim–50–3_4–yes1–4, MO-BB$_{\texttt{icf}}$ is up to 4 times faster than MO-BB$_{fdac}$.

Dubois instances and *pret60_75* require additional discussion. They can be considered as *degenerated* multi-objective instances because their efficient frontier has size 1. It is important to note that we only know that after solving the instances. As noted in Property 4.3.2, when the efficient frontier has only one efficient cost vector, each component of this vector is the optimal cost of each objective function when considered independently. Therefore, they can be solved as two independent mono-objective problems. ϵ-constraint is very efficient on these instances for two reasons. First, the lower bound (l_1, l_2) received by the algorithm is already the efficient frontier. Therefore,

Instance	nb. vars	nb. clauses	$\|\mathcal{E}\|$	Time (sec.)		
				mo-bb$_{\texttt{icf}}$	ϵ-constraint	mo-bb$_{fdac}$
dubois20	60	160	1	125.98	0.002	34.87
dubois21	63	168	1	260.11	0.003	72.01
dubois22	66	176	1	543.91	0.003	149.16
dubois23	69	184	1	1131.71	0.001	308.26
dubois24	72	192	1	-	0.002	637.2
dubois25	75	200	1	-	0.003	1313.52
pret60_60	60	160	6	630.34	-	1646.27
pret60_75	60	160	1	165.03	0.002	99.03
aim-50-1_6-no-1	50	80	8	48.91	1663.47	39.83
aim-50-1_6-no-2	50	80	10	30.62	1202.1	197.69
aim-50-1_6-no-3	50	80	10	46.95	-	60.71
aim-50-1_6-no-4	50	80	10	18.44	-	112.93
aim-50-1_6-yes1-1	50	80	10	20.89	-	51.55
aim-50-1_6-yes1-2	50	80	8	24.93	-	12.28
aim-50-1_6-yes1-3	50	80	10	24.6	-	25.62
aim-50-1_6-yes1-4	50	80	8	5.74	780.23	9
aim-50-2_0-no-1	50	100	12	67.35	-	319.44
aim-50-2_0-no-2	50	100	10	45.61	-	87.88
aim-50-2_0-no-3	50	100	10	17.06	1603.98	32.08
aim-50-2_0-no-4	50	100	10	25.2	-	70.05
aim-50-2_0-yes1-1	50	100	14	97.99	-	725.26
aim-50-2_0-yes1-2	50	100	12	69.68	-	345.93
aim-50-2_0-yes1-3	50	100	14	72.11	-	443.72
aim-50-2_0-yes1-4	50	100	14	150.78	-	1000.52
aim-50-3_4-yes1-1	50	170	15	71.06	-	211.56
aim-50-3_4-yes1-2	50	170	17	199.91	-	520.62
aim-50-3_4-yes1-3	50	170	19	309.71	-	1535.47
aim-50-3_4-yes1-4	50	170	19	184.12	-	900.66
aim-50-6_0-yes1-1	50	300	27	1475.08	-	-
aim-50-6_0-yes1-2	50	300	26	1525.85	-	-
aim-50-6_0-yes1-3	50	300	23	791.3	-	-
aim-50-6_0-yes1-4	50	300	23	690.42	-	-

Figure 5.6: Experimental results on Max-SAT-ONE problems. Time limit 1800 seconds.

ϵ-constraint reduces to solving one CSP problem. Besides, it turns out that $l_2 = 0$. Therefore, the bounding constraint imposed by ϵ-constraint on the second objective function prohibits any variable to be assigned to 0. This situation is rapidly detected by the CSP solver and all the variables are assigned to 1 almost instantaneously.

N (nb. vars)	E (nb. edges)	$\|\mathcal{E}\|$	mo-bb$_{\texttt{icf}}$		ϵ-constraint		mo-bb$_{fdac}$	
			time (sec.)	%	time (sec.)	%	time (sec.)	%
60	100	7,52	4,8768	100	253,616	100	0,4592	100
70		8,64	43,684	100	1403,49	48	2,4208	100
80		8,04	231,737	100	1786,62	8	9,6908	100
90		8,72	984,612	91,6	1800	0	32,7044	100
60	250	5,04	1,7016	100	24,0557	100	0,3276	100
70		6,68	16,0724	100	393,654	100	1,6968	100
80		8,6	110,581	100	1723,12	8	10,6772	100
90		8,76	698,592	96	1800	0	53,9536	100
60	500	4,72	0,3684	100	2,99779	100	0,112	100
70		5,4	3,0384	100	36,263	100	0,5588	100
80		6,6	16,9736	100	323,253	100	2,932	100
90		7,76	117,463	100	1744,17	12	15,7836	100
60	950	3,76	0,082	100	0,33099	100	0,0348	100
70		4,04	0,4992	100	1,78573	100	0,1376	100
80		5,92	2,0096	100	20,434	100	0,5384	100
90		5,56	11,5052	100	126,26	100	2,634	100

Figure 5.7: Experimental results on biobjective weighted minimum vertex cover problems. Parameter C is set to 5. Mean values on 25 instances for each parameter configuration. Time limit 1800 seconds.

5.3.2 Biobjective Minimum Weighted Vertex Cover

Figure 5.7 reports the results obtained in the biobjective minimum weighted vertex cover. The first and second columns show the number of variables and edges, respectively. The third column contains the mean size of the efficient frontier for each class of problem. The remaining columns report the mean cpu time and the percentage of solved instances within the time limit for the three studied algorithms.

MO-BB$_{fdac}$ is the best option for all parameter configurations, while ϵ-constraint is the worst approach. The poor performance of MO-BB$_{\texttt{icf}}$ with respect to MO-BB$_{fdac}$ is hardly a surprise. The structure of the problem leads the lower bound frontier $\texttt{lbf}_{\texttt{icf}}$ to be very loose, as shown in the following example.

Consider the graph in Figure 5.8 (a). It has three nodes $\{x, y, z\}$. Each node has two associated weights, depicted underneath it. Figure 5.8 (b)

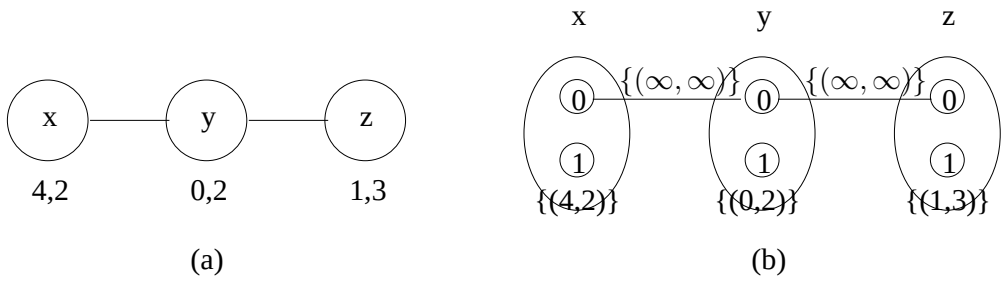

Figure 5.8: (a) A graph where each node has two weights. (b) Graphical representation of the biobjective weighted minimum vertex cover problem on the previous graph as a MO-WCSP.

shows the graphical representation as a MO-WCSP of the biobjective minimum weighted vertex cover problem associated to the previous graph. Only frontiers different from $\{(0,0)\}$ are depicted. Let x be the selected variable to be assigned by MO-BB and let us consider the contribution of each unassigned variable to $\texttt{lbf}_{\texttt{icf}}$. Variable z will not contribute to the lower bound because its inconsistency count is $\texttt{icf}_z = \{(0,0)\}$. In general, variables not connected to the assigned variables do not contribute to $\texttt{lbf}_{\texttt{icf}}$. The contribution of variable y depends on the value assigned to x. If x is assigned to 1, then $\texttt{icf}_y = \{(0,0)\}$. Note that in this case $\texttt{lbf}_{\texttt{icf}}$ equals $\texttt{lbf}_s$. If x is assigned to 0, then $\texttt{icf}_y = \{(0,2)\}$ (because the domain value 1 of y is pruned). Note that in this case, $\texttt{lbf}_{\texttt{icf}}$ equals $\texttt{lbf}_s$ when y is assigned to its remaining domain value 0. Namely, $\texttt{lbf}_{\texttt{icf}}$ only adds to $\texttt{lbf}_s$ the costs given by unassigned variables with a single domain value.

On the other hand, FDAC is able to *simplify* the network extracting to the lower bound necessary costs that any variable assignment will cause. For instance, when the graph is acyclic (as in Figure 5.8), the lower bound computed by FDAC is the optimal cost of each objective function when considered independently. Since this lower bound is always tighter than the one computed by $\texttt{lbf}_{\texttt{icf}}$ and it is computed very efficiently, the cpu time spent by MO-BB$_{fdac}$ is lower than the one spent by MO-BB$_{\texttt{icf}}$.

The good results obtained by MO-BB using a lower bound that is mono-objective in nature seems to indicate that these instances can be solved by MO-BB without using very elaborated lower bound frontiers. We will go back to this hypothesis in Chapters 6 and 8.

Finally, the cpu time of each algorithm with respect to each different class of problem follows the same pattern. When fixing the number of constraints and increasing the number of variables, the efficiency of all the approaches decreases. When fixing the number of variables and increasing the number of constraints, the efficiency of all the approaches increases.

5.3.3 Combinatorial Auctions

Figure 5.9 reports the results obtained for risk-conscious auctions instances with 20 (left) and 50 goods (right). We report mean cpu time (top) and mean solved percentage within the time limit (bottom). We consider the time limit as the cpu time for unsolved instances. We do not plot the results of ϵ-constraint for instances with 50 goods because it fails in solving all instances.

ϵ-constraint is clearly the worst approach. It does not solve all the instances of any parameter configuration. Moreover, it is only able to solve some instances with 20 goods and 80 to 100 bids. Its solved percentage for instances with 80 bids is quite low (28%) and it decreases as the number of bids increases. From 100 to 150 bids, its solved percentage is 0.

Regarding MO-BB$_{\texttt{icf}}$ and MO-BB$_{fdac}$ there is no clear winner. Their behaviour depend on the number of goods. For instances with 20 goods, MO-BB$_{fdac}$ is slightly superior than MO-BB$_{\texttt{icf}}$. The percentage of solved instances by MO-BB$_{fdac}$ surpasses the one obtained by MO-BB$_{\texttt{icf}}$. Moreover, the time spent by MO-BB$_{fdac}$ to solve each parameter configuration is also better than that spent by MO-BB$_{\texttt{icf}}$. For instances with 50 goods, the performance of both algorithms is very similar. However, note that MO-BB$_{\texttt{icf}}$ is able to completely solve instances from three configurations more than MO-BB$_{fdac}$ (from 95 to 105 bids). From 80 to 105 bids MO-BB$_{\texttt{icf}}$ is a

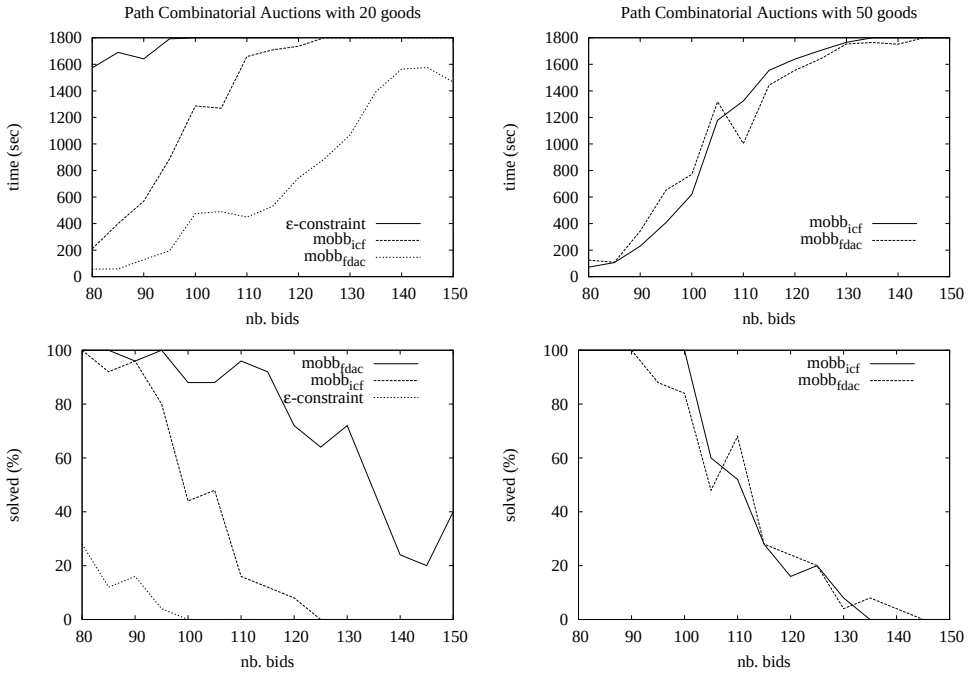

Figure 5.9: Experimental results on risk-conscious combinatorial auctions for 20 and 50 goods, respectively. Path distribution. Mean values on samples of size 25. Time limit 1800 seconds.

little more efficient than MO-BB$_{fdac}$. From 110 to 150 bids is the other way around.

It is important to recall the main difference between MO-BB$_{\texttt{icf}}$ and MO-BB$_{fdac}$. The first algorithm uses a multi-objective lower bound that is quite *naive*. The latter uses a mono-objective lower bound that is very sophisticated and efficient. The results in this benchmark justify the interest of finding wiser multi-objective lower bounds. We will address this issue in Chapter 8.

Instance	# vars	# constrs	Time (sec.)		
			mo-bb$_{\texttt{icf}}$	ϵ-constraint	mo-bb$_{fdac}$
1504(0,183)*	184	1329	-	-	-
1504(184,206)*	23	112	0.36	1246.4	60.2
1504(462,508)*	47	301	-	-	-
1506(0,150)	151	1440	-	-	-
1506(151,228)	78	1107	-	-	-
1506(229,317)	89	1349	-	-	-
1506(679,761)*	83	1243	-	-	-
1405(762,854)*	93	2193	-	-	-
1407(0,147)	148	1442	-	-	-
1407(148,247)	100	1678	-	-	-
1407(248,378)	131	3063	-	-	-
1407(379,409)*	31	220	0.18	-	-
1407(413,429)*	17	87	0.02	16.6	2
1407(447,469)*	23	129	0	16.8	0
1407(494,553)*	60	1333	-	-	-
1407(580,700)	121	2299	-	-	-
1407(701,761)	61	445	-	-	-
1407(762,878)*	117	2708	-	-	-

Figure 5.10: Experimental results on subproblems of the Spot5 instances with capacity constraint. Time limit 1800 seconds.

5.3.4 Scheduling of an EOS

Figure 5.10 reports the results obtained for the scheduling of an EOS benchmark. We break the multiple orbit Spot5 instances into subinstances (as described in Appendix B.2) and run the three algorithms on them. Subinstances which are connected components with less than 20 variables or with trivially satisfiable capacity constraints can be solved almost instantly by any method and we do not report them. The first column of the figure indicates the name of each instance. The second and third columns tell the number of variables and the number of constraints, respectively. The following columns report cpu time in seconds required by each algorithm. Symbol "-" indicates that the algorithm cannot solve the subproblem within the time limit.

The most remarkable observation is that none of the algorithms is able to solve but very few instances. This fact suggests that MO-BB may not be suitable for this kind of instances. Actually, we will see in Chapter 6 that

other multi-objective branch-and-bound algorithm using more sophisticated multi-objective lower bounds is very appropriate in this benchmark.

5.4 Related Work

Other works in the literature explore the extension of the branch-and-bound schema to solve multi-objective optimization problems. One good example is the work by Gavanelli [54], presented in the context of Constraint Programming (CP). As we have seen in Section 3.2.1, mono-objective branch-and-bound is emulated in CP as an iterative process where, at each step, the algorithm solves a CSP problem. Initially, the objective function $F(\mathcal{X})$ is added as a constraint $F(\mathcal{X}) < K$, where K is an upper bound of the optimum. If there exists an assignment X such that the problem is satisfiable, then the algorithm adds a new constraint $F(\mathcal{X}) < F(X)$, and solves it again. This process is iterated until the problem is unsatisfiable. Then, the cost of the last satisfiable assignment is the optimum of the original optimization problem. The parallelism between the updating of the upper bound `ub` in BB and the added constraints is clear. However, the pruning condition and the lower bound are hidden in the internal management (or propagation) of the constraints by the solver.

Gavanelli shows that the multi-objective case can be implemented using similar ideas. Initially, the constraint programming solver adds a set of constraints $F_j(\mathcal{X}) < K_j$, where $F_j(\mathcal{X})$ and K_j are the j^{th} objective function and its upper bound, respectively. Then, the algorithm iterative solves a sequence of CSP problems. At each step, it adds to the previous satisfiable CSP problem a set of constraints that restricts the new solution to be non-dominated with respect the ones previously found. When the CSP problem is unsatisfiable, the set of non-dominated solutions found during this process is the efficient frontier of the original multi-objective problem. Again, there is a parallelism between the new set of constraints and the `ubf` updating. However, it is not easy to see the parallelism between the propagation done by

the CP solver, and the pruning condition and lower bound frontier. In some sense, the CP approach *hides* the main elements of the branch-and-bound schema.

The work of Harikumar et. al in [59] is also related to ours. They extend the *iterative deepening* A^* algorithm to multi-objective optimization. In its description, they propose a multi-objective depth-first branch-and-bound algorithm to compute the efficient solutions of a restricted area of the search space. Its algorithmic structure is quite close to ours. However, it does not explicit a lower bound in the pruning condition. A careful reading of their work shows that their algorithm implicitly uses the simplest lower bound $\texttt{lbf}_s$ to determine whether to continue the search. They do not develop the idea of using bounds to improve the search.

5.5 Conclusions

In this Chapter, we have extended the branch-and-bound schema from mono-objective to multi-objective optimization. The formalization of multi-objective optimization problems as instances of the semiring CSP framework gives us the main elements to extend the three key concepts of branch-and-bound: the upper bound, the lower bound and the pruning condition. As a consequence, the resulting algorithm, called MO-BB, exhibits the same structure as its mono-objective counterpart.

Since the role of upper and lower bounds are clearly defined, it allows MO-BB to use any valid bound without changing its description. We have presented some preliminary upper and lower bound frontiers. These basic bounds are the starting point to more sophisticated multi-objective bounds that will be analyzed in Chapter 6 and Chapter 8.

It is important to note that the generalization of MO-BB to deal with other multi-objective optimization tasks expressed as instances of the semiring CSP framework is straightforward. The only difference being the used of the convenient combination operator and the partial order defined by its

corresponding c-semiring.

Finally, it is also important to recall that branch-and-bound does not take into account the independences among variables, as explained in Section 3.1.1. AND/OR search overcomes this issue by generating the search space following a tree-like ordering of the variables. Initially, AND/OR search was described in the context of mono-objective heuristic search in Artificial Intelligence [106]. Some attempts have been done to extend this approach to multi-objective heuristic search for a particular search space [107]. Recently, [37, 99] describe AND/OR search in the context of graphical models. This approach has been shown very effective for mono-objective optimization tasks [98]. In our current research, we want to investigate the extension of AND/OR search from mono-objective to multi-objective optimization in the context of graphical models.

Chapter 6

Russian Doll Search

Russian doll search (RDS) is a well-known branch-and-bound search schema that solves mono-objective optimization problems by using sophisticated upper and lower bounds. In this Chapter we extend RDS to the multi-objective context. The new algorithm, called *multi-objective russian doll search* (MO-RDS), involves the definition of new valid upper and lower bound frontiers. Our experimental results show that, as RDS, MO-RDS appears to be efficient in problems with relatively small bandwidth. Moreover, this Chapter points out that in some cases it may be convenient to solve mono-objective optimization problems as if they were multi-objective because they can be broken into independent subproblems. We demonstrate this idea by solving for the first time mono-objective SPOT5 instance 1504 using MO-RDS.

The structure of the chapter is as follows. Section 6.1 describes RDS along with its specialized version. Section 6.2 presents the extension of RDS and its specialized version to multi-objective optimization and analyzes the lower and upper bound frontier that they use. Section 6.3 empirically proves the efficiency of the new algorithms in some class of problems. Finally, Section 6.4 gives some conclusions and points out some future work.

6.1 (Mono-objective) Russian Doll Search

Russian doll search (RDS) [145] is a branch-and-bound algorithm which invests in high quality upper and lower bounds. The idea of RDS is to replace one search by n successive searches on nested subproblems, where n is the number of variables in the problem. The first subproblem involves only one variable. Each successive subproblem results from adding one new variable to the previous one. Finally, the last subproblem is the whole problem. Each subproblem is solved to optimality using depth-first branch-and-bound search. The optimal cost of the original problem is obtained with the last search, but the key of the algorithm is that the optimal assignments of the solved subproblems, along with their costs, are used to help subsequent searches. The essence of RDS is to exploit the following straightforward property: the optimal cost of a problem is greater than or equal to the optimal cost of its subproblems.

Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ be a WCSP. Consider an arbitrary static variable ordering, that we assume lexicographic without loss of generality. *Subproblem* P^i is the problem induced by variables $(x_i, \ldots, x_n)$. Formally,

$$P^i = (\mathcal{X}^i, \mathcal{D}^i, \mathcal{F}^i)$$

where

- $\mathcal{X}^i = \{x_i, \ldots, x_n\}$
- $\mathcal{D}^i = \{D_i, \ldots, D_n\}$
- $\mathcal{F}^i = \{f \in \mathcal{F} \mid var(f) \subseteq \mathcal{X}^i\}$

Note that $P^1 = P$. Let X_i^j be a short-hand for an assignment involving variables $(x_i, x_{i+1}, \ldots, x_j)$ and F^i be the objective function of P^i.

RDS sequentially solves subproblems of P, starting with subproblem P^n down to subproblem P^1. Each subproblem P^i is solved to optimality using DF-BB as described in Figure 5.1. The execution of DF-BB for solving P^i is characterized by:

1. An initial upper bound of P^i which is computed from the optimal solution of the previously solved subproblem P^{i+1}.

2. The assignment of variables is done according to the static order of RDS (i.e., $x_i, x_{i+1}, \ldots, x_n$).

3. An specific lower bound $\texttt{lb}_{rds}$ which is used in each search node for pruning purposes.

The optimum of the original problem P (i.e., $opt(P)$) is obtained when subproblem P^1 is solved (because $P^1 = P$).

Consider an execution of RDS. Let P^i be the problem being solved by DF-BB. First, the optimal solution of the previously solved subproblem P^{i+1} can be used to compute an initial upper bound of P^i. The idea is that any extension of an optimal assignment of P^{i+1} to variable x_i is likely to be near-optimal in P^i (because the two problems are very similar), so the best one is used. Note that, since P^{i+1} was solved to optimality in the previous iteration of RDS, its optimal assignment is known when solving P^i. Thus, the initial upper bound of P^i is,

$$\texttt{ub} = \min_{a \in D_i} \{F^i((x_i = a) \cdot t)\}$$

where t is the optimal assignment of P^{i+1}.

Now, consider an arbitrary search node during the resolution of P^i by DF-BB and let X_i^{j-1} be its associated partial assignment. At this point, DF-BB decides whether to prune the current line of search or not on the basis of an underestimation of the current subproblem $P^i(X_i^{j-1})$. As we have seen in Section 5.1.1, DF-BB can use $\texttt{lb}_{ic} = \texttt{lb}_s + \sum_{k=j}^{n} \texttt{ic}_k$ as a lower bound. Since DF-BB assigns the variables in the static order of RDS, it can also use the optimum of the subproblem induced by the current set of unassigned variables $(x_j, \ldots, x_n)$ (i.e., $opt(P^j)$) to improve this bound. Formally,

$$\texttt{lb}_{rds} = \texttt{lb}_s + \sum_{k=j}^{n} \texttt{ic}_k + \texttt{opt}(P^j)$$

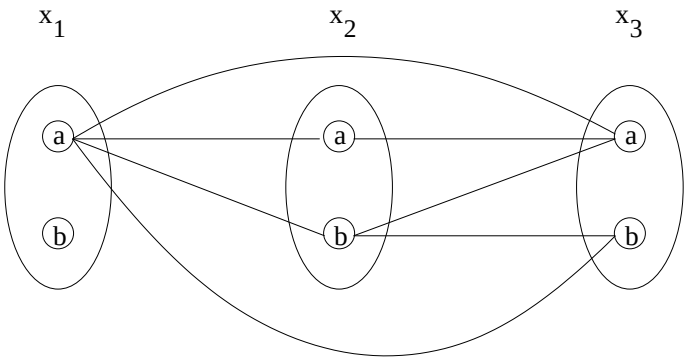

Figure 6.1: WCSP instance.

Note that when solving P^i, $\texttt{opt}(P^j)$ is already known because P^j was solved to optimality in a previous iteration of RDS. As in the original paper [145], we assume that $\mathcal{F}$ in the original problem does not contain unary cost functions. Otherwise, the definition of $\texttt{ic}_{ka}$ needs to be modified in order to disregard original unary functions (because unary functions are already considered in $opt(P^j)$). It is easy to see that $\texttt{lb}_{rds}$ is a lower bound of $P^i(X_i^{j-1})$ because the subset of functions taken into account by each of its three components (i.e., $\texttt{lb}_s$, $\sum_{k=j}^{n} \texttt{ic}_k$ and $\texttt{opt}(P^j)$) are disjunct and each component itself is a lower bound of the optimum of the corresponding subset of functions.

Example 6.1.1 *Consider the WCSP instance of Figure 6.1. It has three variables* $\{x_1, x_2, x_3\}$ *and two domain values per domain* $D_i = \{a, b\}$*. There are three binary cost functions:* $f_{x_1x_2}(x_1, x_2)$*,* $f_{x_1x_3}(x_1, x_3)$ *and* $f_{x_2x_3}(x_2, x_3)$*. Unary costs are* 0*. Binary costs are* 1 *when there exists an edge connecting the corresponding pair of values. Otherwise, the cost is* 0*. Let* $t = (x_1 = a)$ *be the current partial assignment. At this point in search, the lower bound used by BB is as follows. The simplest lower bound is* $\texttt{lb}_s = 0$ *because none of the functions are completely assigned with* t*. There are two unassigned variables:* x_2 *and* x_3*. The inconsistency count of* x_2 *is* $ic_2 = 0$ *because* $f_{x_1x_2}(t \cdot x_2 = a) = 1$ *and* $f_{x_1x_2}(t \cdot x_2 = b) = 0$*. The inconsistency count of* x_3 *is* $ic_3 = 1$ *because* $f_{x_1x_3}(t \cdot x_3 = a) = 1$ *and* $f_{x_1x_3}(t \cdot x_3 = b) = 1$*. The*

function RDS(P) **return** int
1. rds$[n+1] := 0$;
2. **for each** i **from** n **downto** 1 **do**
3. rds$[i]$:=UB(rds$, i+1$);
4. DF-BB(P^i, rds$[i]$);
5. **endfor**
6. **return** rds$[1]$;

endfunction

Figure 6.2: Russian Doll Search algorithm. The input of the algorithm is a WCSP instance $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$. The output of the algorithm is its optimal cost.

optimal cost of subproblem P^2 *is* $opt(P^2) = 0$ *(the optimal assignment is* $(x_2 = a, x_3 = b)$*). The lower bound considered at the current search node is* $lb_{rds} = 0 + (1 + 0) + 0 = 1$.

Figure 6.2 shows an algorithmic description of RDS. The input of the algorithm is the WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ to be solved. It uses an array of costs rds. Cost rds$[i]$ plays the role of the upper bound ub during the resolution of subproblem P^i, so at the end of its resolution it will contain its optimum $opt(P^i)$. For algorithmic convenience, we define rds$[n+1] = 0$. The algorithm solves P^i in decreasing order of i (line 2). First, rds$[i]$ is initialized with a valid upper bound (line 3). Then, P^i is solved with a call to DF-BB (line 4), which assigns variables in the order of RDS and uses the lower bound $\texttt{lb}_{rds}$. When RDS solves the last subproblem P^1, its optimal cost stored in $rds[1]$ is the optimum of P.

6.1.1 Specialized RDS

In RDS, two consecutive subproblems (e.g., P^i and P^{i+1}) differ in one variable only. Each iteration of RDS can be seen as the computation of the cost of

including that new variable in the previously solved subproblem. Sometimes, it may be convenient to compute the cost of including each domain value of that variable in the previously solved subproblem. This new approach, called *Specialized RDS* (SRDS) [103], performs RDS specialized per domain value. The main motivation is that including the domain values of one variable in the previously solved subproblem is not necessaryly homogeneous, that is, *good* domain values (with low cost) and *bad* domain values (with high cost) may be present in the variable domain. Using this specialized contribution, SRDS is able to develop pruning conditions stronger than RDS ones.

SRDS performs up to $n \times d$ independent searches (where n is the number of variables and d the maximum domain size), one for including every domain value of every new variable. Let P^{ia} be a short-hand for $P^i(x_i = a)$, that is, subproblem P^i conditioned to assignment $(x_i = a)$. Each subproblem P^{ia} is solved to optimality with DF-BB. The execution of DF-BB for solving P^{ia} is characterized by:

1. An initial upper bound of P^{ia} computed from the best extension to $x_i = a$ of the optimal assignments of $P^{i+1,b}$ for all $b \in D_{i+1}$.

2. The assignment of variables according to the static order of RDS.

3. The use of an specific lower bound $\texttt{lb}_{srds}$ in each search node.

The optimum of P^i is the best alternative among $opt(P^{ia})$, for all $a \in D_i$. Formally, $opt(P^i) = \min_{a \in D_i}\{opt(P^{ia})\}$. Then, it is clear that the optimum of P is $opt(P) = \min_{a \in D_1}\{opt(P^{1a})\}$.

Consider an execution of SRDS. Let P^{ia} be the problem to be solved by DF-BB. First, the optimal solutions of the previously solved problems $P^{(i+1)b}$ can be used to compute an initial upper bound of P^{ia}. The minimum among the cost of extending the optimal assignment of each subproblem $P^{(i+1)b}$ to $x_i = a$ is an upper bound of $P^i(x_i = a)$. Formally,

$$\texttt{ub}^{ia} = \min_{b \in D_{i+1}}\{F^i(t_b \cdot x_{i+1} = b)\}$$

where t_b is the optimal assignment of problem $P^{i+1}(x_{i+1} = b)$. Note that this upper bound takes advantage of the specialization of P^{i+1} on each of its domain values $b \in D_{i+1}$.

Now, consider an arbitrary search node during the resolution of P^{ia} by DF-BB and let X_i^{j-1} be the current partial assignment (note that $x_i = a$). Since DF-BB assigns variables in the static order of SRDS (i.e., $(x_i, x_{i+1}, \ldots, x_n)$), it can underestimate $P^{ia}(X_i^{j-1})$ using the following lower bound,

$$\texttt{lb}_{srds} = \texttt{lb}_s + \sum_{k=j+1}^{n} \texttt{ic}_k + \min_{b \in D_j}\{\texttt{ic}_{jb} + opt(P^{jb})\}$$

It is easy to see that $\texttt{lb}_{rds} \leq \texttt{lb}_{srds}$.

Example 6.1.2 *Consider the WCSP instance of Example 6.1.1, being $t = (x_1 = a)$ the current partial assignment. At this point in search, the specialized lower bound used by BB is as follows. As before, the simplest lower bound is 0 and the inconsistency count of x_3 is $ic_3 = 1$. The inconsistency counts of the next variable to be assigned (i.e., x_2) are specialized for each of its domain values. Then, $ic_{2a} = 1$, and $ic_{2b} = 0$. The optimal cost of subproblem P^2 is also specialized for each of the domain values of x_2. Then, $opt(P^{2a}) = 0$ and $opt(P^{2b}) = 1$. The lower bound considered at the current search node is $lb_{srds} = 0+1+\min\{1+0, 0+1\} = 2$. Note that in this example $lb_{rds} < lb_{srds}$.*

Figure 6.3 shows a basic description of SRDS[1]. The input of the algorithm is the WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ to be solved. Its structure is very similar to RDS, but specialized to each domain value. Now, `rds` is a matrix of costs where $\texttt{rds}[i, a]$ is the optimal cost of problem P^{ia}. Cost $\texttt{rds}[i, a]$ plays the role of the `ub` when solving P^{ia}, so at the end of its resolution $rds[i, a] = opt(P^{ia})$. For convenience, we suppose that the problem has a dummy variable x_{n+1} with just one domain value 0 and initialize $\texttt{rds}[n +$

[1]The original algorithmic description includes some subtle technical improvements. For clarity reasons, we describe the basic structure only.

function SRDS(P) **return** int
1. rds$[n+1,0] := 0$;
2. **for each** i **from** n **downto** 1 **do**
3. **for each** $a \in D_i$ **do**
4. rds$[i,a]$:=UB(rds$, i+1, a$);
5. DF-BB(P^{ia}, rds$[i,a]$);
6. **endfor**
7. **endfor**
8. **return** $\min_{a \in D_1}$\{rds$[1,a]$\};

endfunction

Figure 6.3: Specialized Russian Doll Search algorithm.

$1,0] = 0$ (line 1). SRDS solves P^i in decreasing order of i and for each of its domain values $a \in D_i$ (line 2-3). First, $rds[i,a]$ is initialized with a valid upper bound (line 4). Then, $P^i(x_i = a)$ is solved by DF-BB (line 5) using the static order of SRDS and the lower bound lb_{srds}. When SRDS solves the last subproblem P^1, the minimum among the optimums of problem P^1 specialized to its domain values $a \in D_1$ is the optimum of P (line 8).

6.2 Multi-objective Russian Doll Search

Multi-objective optimization problems can also be solved using a russian doll search schema. Let P be a MO-WCSP problem. The algorithm computes the efficient frontier of P by n successive searches on nested subproblems from P^n down to P^1. Now, the result of solving subproblem P^i is its efficient frontier $\mathcal{E}(P^i)$. The output of the algorithm is the efficient frontier $\mathcal{E}(P^1)$, that is, the efficient frontier of the original problem P.

As before, the key of the algorithm is that information from previous resolutions is used to compute new upper and lower bounds (in this case frontiers) that help to solve subsequent subproblems. It is based in Property

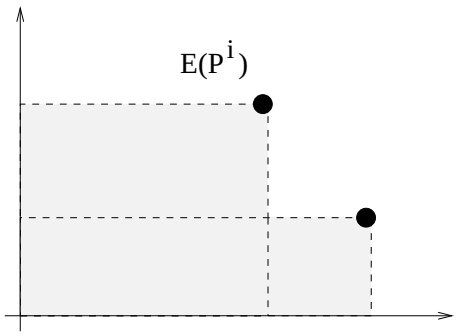

Figure 6.4: Efficient frontier of a bi-objective optimization problem. Vectors in the gray rectangle will dominate the vector in its top-right corner.

4.3.5, which in this context states that: given a problem P^i, its efficient frontier $\mathcal{E}(P^i)$ is dominated by the efficient frontier $\mathcal{E}(P^j)$ of any subproblem P^j. Formally,

$$\forall\, 1 \leq i \leq j \leq n,\ \mathcal{E}(P^i) \geq_{mo} \mathcal{E}(P^j)$$

This Property, which was straightforward in the mono-objective case, may not be so direct when considering two or more objectives. We illustrate it by means of an example. Let P^i be a bi-objective problem and let the frontier of Figure 6.4 be its efficient frontier $\mathcal{E}(P^i)$. Each cost vector $\vec{v} \in \mathcal{E}(P^i)$ is the valuation of one complete assignment in the objective function F^i. Let $X(\vec{v})$ be the complete assignment such that $F^i(X(\vec{v})) = \{\vec{v}\}$. Each vector $\vec{v} \in \mathcal{E}(P^i)$ is the top right corner of a gray rectangle. Let P^j be a subproblem of P^i. Since $\mathcal{F}^j \subseteq \mathcal{F}^i$, then $F^i = F^j + \sum_{f \in \mathcal{F}^i - \mathcal{F}^j} f$. It is clear that the valuation of $X(\vec{v})$ in F^j is a cost vector contained in the rectangle associated to $\vec{v}$. Namely, $F^j(X(\vec{v})) = \{\vec{w}\}$ such that $\vec{w} \leq \vec{v}$. If $\vec{w} \in \mathcal{E}(P^j)$, then it dominates $\vec{v}$. Otherwise, it would mean that there is another vector $\vec{u} \in \mathcal{E}(P^j)$ that dominates $\vec{w}$, and as a consequence, $\vec{u}$ also dominates $\vec{v}$. As a result, $\forall\, \vec{v} \in \mathcal{E}(P^i),\ \exists\, \vec{u} \in \mathcal{E}(P^j)$ such that $\vec{u} \leq \vec{v}$, namely $\mathcal{E}(P^j) \leq_{mo} \mathcal{E}(P^i)$.

Multi-objective russian doll search (MO-RDS) sequentially solves subproblems P^i in decreasing order of i. Each subproblem P^i is solved with

a call to MO-BB (described in Figure 5.4). Each execution of MO-BB is characterized by:

1. An initial upper bound frontier computed from the efficient solutions of the previously solved subproblem.

2. The assignment of variables in the static order of MO-RDS.

3. The use of the lower bound frontier $\texttt{lbf}_{rds}$ in each search node.

The efficient frontier of the original problem P is obtained with the last search.

Consider an execution of MO-RDS. Let P^i be the subproblem being solved by MO-BB. In the following, we define the initial upper bound frontier and the lower bound frontier used by MO-BB.

Initial upper bound frontier

As noted in classical RDS, the resolution of P^{i+1} contains useful information to compute an initial upper bound frontier of P^i. Now, the best extensions of the efficient solutions of P^{i+1} form an upper bound frontier of P^i. Formally, let $\vec{v} \in \mathcal{E}(P^{i+1})$ and let $X_{i+1}^n(\vec{v})$ be the optimal assignment of P^{i+1} such that $F^{i+1}(X_{i+1}^n(\vec{v})) = \{\vec{v}\}$. We define

$$W_{\vec{v}} = \bigcup_{a \in D_i} \{F^i((x_i = a) \cdot X_{i+1}^n(\vec{v}))\}$$

(i.e., the set of extensions of one efficient solution of P^{i+1}). Then, `ubf` is the set of best cost vectors among all sets $W_{\vec{v}}$,

$$\texttt{ubf} = \min_{\vec{v} \in \mathcal{E}(P^{i+1})}^{mo} \{W_{\vec{v}}\}$$

Theorem 6.2.1 *`ubf` is a valid upper bound frontier of problem P^i.*

Proof The theorem is an obvious consequence of Property 4.3.4. By definition, `ubf` is a set of non-dominated elements coming from a subset of

complete assignments (the set of efficient solutions of $\mathcal{E}(P^{i+1})$ extended to variable x_i), while $\mathcal{E}(P^i)$ is a set of non-dominated elements coming from all complete assignments.

Lower bound frontier

Consider an arbitrary search node during the resolution of P^i by MO-BB such that X_i^{j-1} is its current partial assignment. As we have seen in Section 5.2.1, MO-BB can use the lower bound frontier $\texttt{lbf}_{ic} = \texttt{lbf}_s + \sum_{k=i}^{n} \texttt{icf}_k$ to underestimate $\mathcal{E}(P^i(X_i^{j-1}))$. Now, since MO-BB assigns the variables in the order of MO-RDS, the algorithm can improve this bound by adding the efficient frontier of subproblem P^j. Formally,

$$\texttt{lbf}_{rds} = \texttt{lbf}_s + \sum_{k=j}^{n} \texttt{icf}_k + \mathcal{E}(P^j)$$

Theorem 6.2.2 *$\texttt{lbf}_{rds}$ is a valid lower bound frontier of problem P^i.*

Proof Let P^i be the subproblem being solved by MO-RDS. Consider an arbitrary search node such that X_i^{j-1} is the current assignment. Thus, $P^i(X_i^{j-1})$ is the subproblem associated to the current search node. For clarity reasons, $P^i(X_i^{j-1})$ will be called P'. Let $\mathcal{F}'$ be the set of functions of P' and $\mathcal{E}(P')$ be its efficient frontier. We have to prove that $\texttt{lbf}_{rds} \leq_{mo} \mathcal{E}(P')$.

The set of functions $\mathcal{F}'$ can be partitioned into three sets: completely assigned functions in P', partially assigned functions in P^i by X_i^{j-1}, and functions in P^i that has not been modified by X_i^{j-1} (i.e., functions of subproblem P^j). By Theorem 4.3.4,

$$\sum_{\substack{f \in P' \\ |f|=0}} f() + \min_{X \in l(\mathcal{X}')}^{mo} \{(\sum_{\substack{f \in P' \\ f \notin P^j}} f)(X)\} + \mathcal{E}(P^j) \leq_{mo} \mathcal{E}(P')$$

which can be rewritten as

$$\texttt{lbf}_s + \min_{X \in l(\mathcal{X}')}^{mo} \{(\sum_{\substack{f \in P' \\ f \notin P^j}} f)(X)\} + \mathcal{E}(P^j) \leq_{mo} \mathcal{E}(P') \tag{6.1}$$

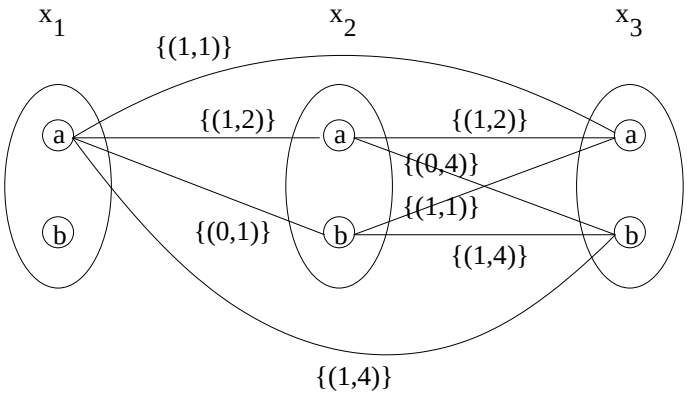

Figure 6.5: MO-WCSP instance.

By definition of $\texttt{icf}_k$, $\sum_{k=j}^{n} \texttt{icf}_k$ is the efficient frontier of a problem that only contains unary functions from the set $\{f \in P' \mid f \notin P^j\}$. Then, by Property 4.3.5,

$$\sum_{k=j}^{n} \texttt{icf}_k \leq_{mo} \min_{X \in l(\mathcal{X}')}^{mo} \{(\sum_{\substack{f \in P' \\ f \notin P^j}} f)(X)\}$$

By monotonicity of $\leq_{mo}$ on the previous expression,

$$\texttt{lbf}_s + \sum_{k=j}^{n} \texttt{icf}_k + \mathcal{E}(P^j) \leq_{mo} \texttt{lbf}_s + \min_{X \in l(\mathcal{X}')}^{mo} \{(\sum_{\substack{f \in P' \\ f \notin P^j}} f)(X)\} + \mathcal{E}(P^j) \quad (6.2)$$

Finally, by transitivity of $\leq_{mo}$ on (6.2) and (6.1),

$$\texttt{lbf}_s + \sum_{k=j}^{n} \texttt{icf}_k + \mathcal{E}(P^j) \leq_{mo} \mathcal{E}(P')$$

Example 6.2.1 *Consider the MO-WCSP instance of Figure 6.5. It is based on the WCSP of Example 6.1 adding one new objective. The problem has three variables* $\{x_1, x_2, x_3\}$ *and two domain values per domain* $D_i = \{a, b\}$. *There are three binary frontier functions:* $f_{x_1x_2}(x_1, x_2)$, $f_{x_1x_3}(x_1, x_3)$ *and*

function MO-RDS(P) **return** frontier
1. rdsf$[n+1] := \{\vec{0}\}$;
2. **for each** i **from** n **downto** 1 **do**
3. rdsf$[i]$:=UBF(rdsf$, i+1$);
4. MO-BB(P^i, rdsf$[i]$);
5. **endfor**
6. **return** rdsf[1];

endfunction

Figure 6.6: Multi-objective Russian Doll Search algorithm. The input of the algorithm is a MO-WCSP instance $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$. The output of the algorithm is its efficient frontier $\mathcal{E}(P)$.

$f_{x_2x_3}(x_2, x_3)$. *Unary valuations are* $\{(0,0)\}$. *Binary valuations are depicted as vector labeled edges connecting the corresponding pair of values. Only non-zero frontiers are shown. Let* $t = (x_1 = a)$ *be the current partial assignment. At this point in search, the simplest lower bound frontier is* $\mathtt{lbf}_s = \{(0,0)\}$ *because none of the frontier functions are totally assigned by* t. *There are two unassigned variables* x_2 *and* x_3. *The inconsistency count of* x_2 *is* $\mathtt{icf}_2 = \{(0,1)\}$ *because* $f_{x_1x_2}(t \cdot x_2 = a) = \{(1,2)\}$, $f_{x_1x_2}(t \cdot x_2 = b) = \{(0,1)\}$, *and* $\mathtt{icf}_2 = \min^{mo}\{\{(1,2)\},\{(0,1)\}\}$. *The inconsistency count of* x_3 *is* $\mathtt{icf}_3 = \{(1,1)\}$ *because* $f_{x_1x_3}(t \cdot x_3 = a) = \{(1,1)\}$, $f_{x_1x_3}(t \cdot x_3 = b) = \{(1,4)\}$, *and* $\mathtt{icf}_3 = \min^{mo}\{\{(1,1)\},\{(1,4)\}\}$. *The efficient frontier of subproblem* P^2 *is* $\mathcal{E}(P^2) = \{(0,4),(1,1)\}$ *with efficient solutions* $(x_2 = a, x_3 = b)$ *and* $(x_2 = b, x_3 = a)$, *respectively. The lower bound frontier used by MO-BB at the current search node is* $\mathtt{lbf}_{rds} = \{(0,0)\} + \{(0,1)\} + \{(1,1)\} + \{(0,4),(1,1)\} = \{(1,6),(2,3)\}$.

Algorithmic description

Figure 6.6 is an algorithmic description of MO-RDS. The input of the algorithm is a MO-WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$. It uses an *array of*

frontiers `rdsf`. Frontier `rdsf`$[i]$ plays the role of the upper bound frontier `ubf` during the resolution of subproblem P^i, so at the end of its resolution it will contain its efficient frontier $\mathcal{E}(P^i)$. The algorithm solves P^i in decreasing order of i (line 2). First, `rdsf`$[i]$ is initialized with a valid upper bound frontier (line 3). Then, each subproblem P^i is solved with a call to MO-BB (line 4). As we have seen, the important features of MO-BB are that it uses the static order of MO-RDS to assign variables and the lower bound frontier $\texttt{lbf}_{rds}$. The efficient frontier of the original problem P is obtained when the last subproblem P^1 is solved. Thus, `rdsf[1]` is the efficient frontier of P.

Property 6.2.1 *When considering a WCSP problem (i.e., $p = 1$), the algorithm MO-RDS is equivalent to RDS.*

6.2.1 Specialized MO-RDS

As shown in the mono-objective case, it may be convenient to ask russian doll to solve more subproblems in order to obtain more accurate bounds. This idea can also be applied to multi-objective optimization.

Specialized MO-RDS (SMO-RDS) solves P^{ia} for each variable i from n down to 1 and for each domain value $a \in D_i$. Each subproblem P^{ia} is solved to optimality by MO-BB, with the following features:

1. The initial upper bound frontier of P^{ia} is computed from the efficient solutions of the previously solved subproblems P^{i+1} conditioned to each domain value $b \in D_{i+1}$.

2. The assignment of variables follows the static order of SMO-RDS.

3. A specific lower bound frontier $\texttt{lbf}_{srds}$ is used in each search node.

The efficient frontier of the original MO-WCSP problem P is obtained when problem P^{1a} has been solved for all $a \in D_1$. Vectors in the efficient frontier of problem P^{1a} represent the best alternatives when variable x_1 is assigned to

domain value a. Then, the best alternatives among all possible assignments of variable x_1 is $\mathcal{E}(P) = \min_{a \in D_1}^{mo}\{\mathcal{E}(P^{1a})\}$.

Consider an execution of SMO-RDS. Let P^{ia} be the problem being solved by MO-BB. Before solving it, SMO-RDS computes an initial upper bound frontier of the subproblem, following the same idea as in MO-RDS. Each efficient solution of $P^{(i+1)b}$ is extended to variable x_i assigned to a. Formally, let $\vec{v} \in \mathcal{E}(P^{(i+1)b})$ and let $X_{i+1}^n(\vec{v}))$ be an optimal assignment with respect to $P^{(i+1)b}$ such that $F^{(i+1)b}(X_{i+1}^n) = \{\vec{v}\}$. We define

$$W_b = \bigcup_{\vec{v} \in \mathcal{E}(P^{i+1,b})} \{F^i((x_i = a) \cdot X_{i+1}^n(\vec{v}))\}$$

(i.e., the set of extensions of subproblem $P^{i+1,b}$). Then, $\texttt{ubf}^{ia}$ is the set of best cost vectors among all W_b,

$$\texttt{ubf}^{ia} = \min_{b \in D_{i+1}}^{mo}\{W_b\}$$

Theorem 6.2.3 *$\texttt{ubf}^{ia}$ is a valid upper bound frontier of problem P^{ia}.*

Proof The proof follows the same structure as Theorem 6.2.1.

Now, consider an arbitrary search node during the resolution of P^{ia} by MO-BB and let X_i^{j-1} be the current partial assignment. Then, the lower bound frontier used by MO-BB is,

$$\texttt{lbf}_{srds} = \texttt{lbf}_s + \sum_{k=j+1}^{n} \texttt{icf}_k + \min_{b \in D_j}^{mo}\{\texttt{icf}_{jb} + \mathcal{E}(P^{jb})\}$$

Theorem 6.2.4 *$\texttt{lbf}_{srds}$ is a valid lower bound frontier of problem P^{ia}.*

Proof The structure of the proof is the same as for Theorem 6.2.2. In this case, $P' = P^{ia}(X_i^{j-1})$ and its set of functions $\mathcal{F}'$ is partitioned into three sets: completely assigned functions in P', partially assigned functions in P^{ia} by X_i^{j-1} except unary functions with scope $\{x_j\}$, and functions in P^{ia} that

has not been modified by X_i^{j-1} (i.e., functions of subproblem P^j) plus unary functions with scope $\{x_j\}$. We call P^{j+} the problem induced by the last subset of functions.
The proof relies in that: by Property 4.3.5

$$\sum_{k=j+1}^{n} \texttt{icf}_k \leq_{mo} \min_{X \in l(\mathcal{X}')}^{mo} \{(\sum_{\substack{f \in P' \\ f \notin P^j \\ x_k \notin var(f)}} f)(X)\}$$

and

$$\min_{b \in D_j}^{mo} \{\texttt{icf}_{jb} + \mathcal{E}(P^{jb})\} = \mathcal{E}(P^{j+})$$

The following theorem shows that the lower bound frontier of SMO-RDS is always tighter than the lower bound of MO-RDS.

Theorem 6.2.5 $\texttt{lbf}_{rds} \leq_{mo} \texttt{lbf}_{srds}$.

Proof Let P^i be the problem being solved by DF-BB, and let X_i^{j-1} be the current partial assignment. Then, by definition of $\texttt{lbf}_{rds}$ and $\texttt{lbf}_{srds}$, we must prove that:

$$\sum_{k=j+1}^{n} \texttt{icf}_k + \texttt{icf}_j + \mathcal{E}(P^j) \leq_{mo} \sum_{k=j+1}^{n} \texttt{icf}_k + \min_{b \in D_j}^{mo} \{\texttt{icf}_{jb} + \mathcal{E}(P^{jb})\}$$

Observe that:

$$\min_{b \in D_j}^{mo} \{\texttt{icf}_{jb} + \mathcal{E}(P^{jb})\} = \mathcal{E}(P^{j+})$$

where P^{j+} is a problem containing all the functions of problem P^j plus unary functions in $P^i(X_i^{j-1})$ with scope $\{x_j\}$. Moreover, $\texttt{icf}_j$ is the efficient frontier of a problem containing only unary functions in $P^i(X_i^{j-1})$ with scope $\{x_j\}$. Then, by Theorem 4.3.4,

$$\texttt{icf}_j + \mathcal{E}(P^j) \leq_{mo} \min_{b \in D_j}^{mo} \{\texttt{icf}_{jb} + \mathcal{E}(P^{jb})\}$$

By monotonicity of $\leq_{mo}$,

$$\sum_{k=j+1}^{n} \texttt{icf}_k + \texttt{icf}_j + \mathcal{E}(P^j) \leq_{mo} \sum_{k=j+1}^{n} \texttt{icf}_k + \min_{b \in D_j}^{mo} \{\texttt{icf}_{jb} + \mathcal{E}(P^{jb})\}$$

Example 6.2.2 *Consider the MO-WCSP instance of Example 6.2.1, being* $t = (x_1 = a)$ *the current partial assignment. At this point in search, the specialized lower bound used by MO-BB is as follows. As before, the simplest lower bound frontier* $\texttt{lbf}_s$ *is* $\{(0,0)\}$ *and the inconsistency counts of* x_3 *is* $\texttt{icf}_3 = \{(1,1)\}$*. The inconsistency counts of the next variable to be assigned (i.e.,* x_2*) is specialized for each of its domain values. Then,* $\texttt{icf}_{2a} = \{(1,2)\}$*, and* $\texttt{icf}_{2b} = \{(0,1)\}$*. The efficient frontier of subproblem* P^2 *is also specialized for each of the domain values of* x_2*. Then,* $\mathcal{E}(P^{2a}) = \{(1,2),(0,4)\}$ *and* $\mathcal{E}(P^{2b}) = \{(1,1)\}$*. The lower bound frontier considered at the current search node is* $\texttt{lbf}_{srds} = \{(0,0)\} + \{(1,1)\} + \min^{mo}\{\{(1,2)\} + \{(0,4),(1,1)\},\{(0,1)\} + \{(1,1)\}\} = \{(2,3)\}$*. Note that in this example* $\texttt{lbf}_{rds} <_{mo} \texttt{lbf}_{srds}$*.*

Figure 6.7 describes SMO-RDS. The input of the algorithm is the MO-WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ to be solved. It uses a matrix of efficient frontiers `rdsf`, where $\texttt{rdsf}[i,a]$ is the efficient frontier of problem P^{ia}. Frontier $\texttt{rdsf}[i,a]$ plays the role of the `ubf` when solving P^{ia}, so at the end of its resolution $\texttt{rdsf}[i,a] = \mathcal{E}(P^{ia})$. For convenience, `rdsf` has to be defined in position $n+1$. We suppose that there exists a dummy variable x_{n+1} with just one domain value 0 and initialize $\texttt{rdsf}[n+1,0]$ to the unit element $\{\overrightarrow{0}\}$ of the combination operator (line 1). The algorithm solves P^{ia} in decreasing order of i and for each domain value $a \in D_i$ (lines 2-3). First, $\texttt{rdsf}[i,a]$ is initialized with a valid upper bound frontier (line 3). Then, each P^{ia} is optimally solved by MO-BB (line 5) using the static order of SMO-RDS to assign its variables and the lower bound frontier $\texttt{lbf}_{srds}$. Finally, the efficient frontier of the overall problem is obtained as $\min^{mo}_{a \in D_1}\{\texttt{rdsf}[1,a]\}$ (line 9).

function SMO-RDS(P) **return** frontier
1. rdsf$[n+1,0] := \{\vec{0}\}$;
2. **for each** i **from** n **downto** 1 **do**
3. **for each** $a \in D_i$ **do**
4. rdsf$[i,a]$:=UBF(rdsf$, i+1, a$);
5. MO-BB(P^{ia}, rdsf$[i,a]$);
6. **endfor**
7. **endfor**
8. **return** $\min_{a \in D_1}^{mo}\{$rdsf$[1,a]\}$;

endfunction

Figure 6.7: Multi-objective Specialized Russian Doll Search algorithm. The input of the algorithm is a MO-WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$. The output of the algorithm is its efficient frontier.

6.3 Experimental Results

The purpose of these experiments is to evaluate the suitability of multi-objective russian doll algorithms for solving MO-WCSP problems. This evaluation analyzes three aspects: (i) the performance of MO-RDS and SMO-RDS with respect to other multi-objective algorithms; (ii) the performance of MO-RDS with respect to SMO-RDS; and (iii) the performance of both russian doll algorithms with respect to the graph bandwidth of the problem.

We compare the performance of MO-RDS and SMO-RDS versus the best approach from the previous Chapter. Namely, the best out of:

- MO-BB$_{\texttt{icf}}$.
- ϵ-constraint.
- MO-BB$_{fdac}$.

We test their performance on the same benchmarks as in the previous Chapter. Namely, *Max-SAT-ONE*, *biobjective minimum vertex cover*, *risk-*

conscious combinatorial auctions and *scheduling of an EOS* benchmarks (for a detailed description of the benchmarks see Appendices B.3, B.4, B.1 and B.2, respectively).

As shown in the mono-objective case [145], russian doll algorithms are proved to be efficient in problems with relatively small bandwidth. To assess the influence of the graph bandwidth on the relative performance of multi-objective russian doll algorithms, we experiment with different variable orderings. In particular, we run MO-RDS and SMO-RDS using the lexicographical ordering and using the variable ordering leading to the smallest graph bandwidth (in each particular benchmark) considering the following heuristics:

- Greedy min-fill heuristic [35].
- Minimal width order (MWO) [50].
- Minimal triangulation (LEX-M) [125].

Moreover, we go one step further and show that a high graph bandwidth does not always mean that russian doll algorithms are not convenient. Note that, since the graph bandwidth is the maximum among the bandwidth of each node of the graph, a high graph bandwidth only means that there exists at least one node with that bandwidth. However, the intuition is that if very few constraints are responsible of a high bandwidth, russian doll methods may still be efficient. We give a first insight on this claim by executing SMO-RDS on special instances from the *biobjective minimum vertex cover* benchmark.

The time limit for all the experiments is 1800 seconds.

6.3.1 Max-SAT-ONE

Figure 6.8 reports the results of the Max-SAT-ONE benchmark. The first and second columns contain the name of the instance and the size of the efficient frontier. The third column shows the graph bandwidth of each instance using

Instance	$\|\mathcal{E}\|$	Lex / LEX-M			mo–bb$_{\texttt{icf}}$	ϵ–constr.	mo–bb$_{fdac}$
		band.	mo–rds	smo–rds			
dubois20	1	41 / 4	543.24 / 116.95	- / 143.78	125.98	0.002	34.87
dubois21	1	43 / 4	1192.67 / 226.34	- / 279.28	260.11	0.003	72.01
dubois22	1	45 / 4	- / 499.57	- / 623.57	543.91	0.003	149.16
dubois23	1	47 / 4	- / 956.7	- / 1200.17	1131.71	0.001	308.26
dubois24	1	49 / 4	- / -	- / -	-	0.002	637.2
dubois25	1	51 / 4	- / -	- / -	-	0.003	1313.52
pret60_40	14	56 / 11	- / 457.28	- / 641.44	-	-	-
pret60_60	6	56 / 11	- / 209.67	- / 255.34	630.34	-	1646.27
pret60_75	1	56 / 11	1540.85 / 215.57	- / 248.47	165.03	0.002	99.03
aim-50-1_6-no-1	8	49 / 35	- / -	- / -	48.91	1663.47	39.83
aim-50-1_6-no-2	10	45 / 33	976.29 / -	- / -	30.62	1202.1	197.69
aim-50-1_6-no-3	10	45 / 28	- / -	- / -	46.95	-	60.71
aim-50-1_6-no-4	10	48 / 37	- / -	- / -	18.44	-	112.93
aim-50-1_6-yes1-1	10	49 / 32	1421.85 / 610	-/ -	20.89	-	51.55
aim-50-1_6-yes1-2	8	49 / 35	- / -	- / -	24.93	-	12.28
aim-50-1_6-yes1-3	10	48 / 32	- / 278.34	- / 958.45	24.6	-	25.62
aim-50-1_6-yes1-4	8	49 / 34	- / 498.78	- / -	5.74	780.23	9
aim-50-2_0-no-1	12	47 / 38	- / 1057.38	- / -	67.35	-	319.44
aim-50-2_0-no-2	10	47 / 32	- / 1500.73	- / -	45.61	-	87.88
aim-50-2_0-no-3	10	46 / 40	1273.79 / 530.66	- / 764.73	17.06	1603.98	32.08
aim-50-2_0-no-4	10	48 / 36	- / 1554.22	- / -	25.2	-	70.05
aim-50-2_0-yes1-1	14	48 / 38	- / -	- / -	97.99	-	725.26
aim-50-2_0-yes1-2	12	43 / 44	- / -	- / -	69.68	-	345.93
aim-50-2_0-yes1-3	14	46 / 45	- / -	- / -	72.11	-	443.72
aim-50-2_0-yes1-4	14	45 / 29	- / -	- / -	150.78	-	1000.52
aim-50-3_4-yes1-1	15	48 / 42	- / -	- / -	71.06	-	211.56
aim-50-3_4-yes1-2	17	49 / 42	- / -	- / -	199.91	-	520.62
aim-50-3_4-yes1-3	19	49 / 39	- / -	- / -	309.71	-	1535.47
aim-50-3_4-yes1-4	19	48 / 42	- / -	- / -	184.12	-	900.66
aim-50-6_0-yes1-1	27	49 / 39	- / -	- / -	1475.08	-	-
aim-50-6_0-yes1-2	26	47 / 44	- / -	- / -	1525.85	-	-
aim-50-6_0-yes1-3	23	48 / 43	- / -	- / -	791.3	-	-
aim-50-6_0-yes1-4	23	47 / 42	- / -	- / -	690.42	-	-

Figure 6.8: Experimental results on Max-SAT-ONE problems. Time limit 1800 seconds.

the lexicographical and LEX-M ordering (which lead to the smallest graph bandwidths w.r.t. min-fill and MWO orderings as shown in B.4) separated by a "/". The fourth and fifth columns indicate the cpu time in seconds required by MO-RDS and SMO-RDS for solving each instance using the two

previous orderings. The remaining columns indicate the cpu time required by MO-BB$_{\texttt{icf}}$, ϵ-constraint and MO-BB$_{fdac}$, respectively. A "-" indicates that the algorithm does not terminate within the time limit.

The main thing to be observed is that russian doll search is not very suitable in this benchmark. In general, MO-RDS and SMO-RDS are the worst options. The only exception is in *pret60_40* and *pret60_60* instances, where both russian doll algorithms using LEX-M order are the best approaches.

The previous results are consistent with the hypothesis that russian doll algorithms are not very convenient for problems with relatively large graph bandwidth. Note that the graph bandwidth for all but *dubois* and *pret* instances is almost the number of variables. It is worth noting that *dubois* and *pret* instances are precisely the instances where MO-RDS and SMO-RDS obtain their best relative performance.

The efficiency of MO-RDS and SMO-RDS seems to depend on the graph bandwidth. For each instance, both algorithms are more efficient using the order that leads to the smallest graph bandwidth. The only exception is in *aim-50-1_6-no-2* instance, where MO-RDS using lexicographical ordering outperforms the same algorithm using LEX-M ordering.

Regarding the relative efficiency between MO-RDS and SMO-RDS, it seems that SMO-RDS does not take advantage of its specialized lower bound. This fact suggests that the lower bound of a given subproblem P^i is very similar to the lower bound of each subproblem $P^i(x_i = a)$. As a consequence, MO-RDS outperforms SMO-RDS because the time spent to compute the specialized lower bound does not lead to a great improve in the pruning capabilities.

6.3.2 Biobjective Minimum Vertex Cover

Figure 6.9 reports the results obtained in the biobjective minimum weighted vertex cover benchmark. The first and second columns indicate the number of variables and edges, respectively. The third column reports the mean graph bandwidth of each parameter configuration using the lexicographical

N	E	Lex / Lex-M			
		bandwidth	mo–rds	smo–rds	mo-bb$_{fdac}$
60	100	55.24 / 45.68	213,87 / 240,98	298,91 / 171.66	0,45
70		64.44 / 54.32	1198,08 (56%) / 735,64 (80%)	1472,21 (40%) / 673.33 (80%)	2,42
80		72.52 / 57.52	1639,28 (20%) / 1343,46 (56%)	1770,68 (4%) / 1236.9 (68%)	9,69
90		79.68 / 60.8	1752,61 (8%) / 1367,1 (40%)	1800 (0%) / 1307.42 (48%)	32,70
60	250	56.96 / 47.32	333,16 / 118,3	259,65 / 82.31	0,32
70		66.32 / 58.28	1800 (0%) / 1488,47 (40%)	1800 (0%) / 1284.61 (75%)	1,69
80		75.84 / 66.28	1800 (0%) / 1800 (0%)	1800 (0%) / 1800 (0%)	10,67
90		84.68 / 73.36	1800 (0%) / 1800 (0%)	1800 (0%) / 1800 (0%)	53,95
60	500	57.64 / 49.88	35,34 / 17,54	16,39 / 9.82	0,11
70		67.6 / 58.92	708,33 / 281	385,64 / 161.98	0,55
80		76.72 / 66.68	1800 (0%) / 1716,9 (24%)	1800 (0%) / 1573.32 (36%)	2,93
90		87.12 / 73.92	1800 (0%) / 1800 (0%)	1800 (0%) / 1800 (0%)	15,78
60	950	58.36 / 53.48	2,05 / 1,47	0,74 / 0.57	0,03
70		68 / 61.88	32,92 / 14,88	10,53 / 6.37	0,13
80		77.56 / 69.12	11145,3 / 178,72	157,96 / 77.13	0,53
90		87.32 / 78.08	1800 (0%) / 1697,21 (32%)	1768,4 (12%) / 1075.99 (88%)	2,63

Figure 6.9: Experimental results on biobjective weighted minimum vertex cover problems. Parameter C is set to 5. Mean values on 25 instances for each parameter configuration. Time limit 1800 seconds.

and LEX-M orders. The fourth and fifth columns report the mean cpu time in seconds required by MO–RDS and SMO–RDS for solving each parameter configuration and the percentage of solved instances between parenthesis using the previous variable orderings. The sixth column shows the cpu time required by MO-BB$_{fdac}$ (i.e., the best approach in this benchmark among MO-BB$_{\texttt{icf}}$, ϵ-constraint and MO-BB$_{fdac}$), and the solved percentage between parenthesis when different from 100%.

MO-RDS and SMO-RDS are not very convenient in this benchmark either. MO-BB$_{fdac}$ outperforms MO-RDS and SMO-RDS in all parameter configurations.

These results also support the hypothesis that the graph bandwidth affects the performance of russian doll algorithms. If we compare the performance of MO-RDS and SMO-RDS in each parameter configuration with respect to each variable order, both algorithms obtain their best results with

LEX-M order. Observe that the graph bandwidth using LEX-M order is always the smallest. Therefore, it seems that MO-RDS and SMO-RDS are more efficient in problems with small bandwidth.

6.3.3 Combinatorial Auctions

Figure 6.10 reports the results obtained for risk-conscious auctions instances with 20 (left) and 50 goods (right). We report mean cpu time (top) and mean solved percentage within the time limit (bottom). We consider the time limit as the cpu time for unsolved instances. We disregard the results obtained by ϵ-constraint because, as seen in the previous Chapter, it is outperformed by MO-BB$_{\texttt{icf}}$ and MO-BB$_{fdac}$.

Multi-objective russian doll search algorithms seem to be not very convenient for these instances. One reason could be the high bandwidth of all parameter configurations given by either the lexicographical as the LEX-M order. Among the russian doll algorithms, SMO-RDS using LEX-M variable ordering is the best approach. However, its solved percentage is below the 80% in any case.

If we focus on the relative performance among russian doll algorithms, it seems that the benefit comes from using a variable ordering leading to small bandwidth rather than using an specialized lower bound. Observe that the efficiency of MO-RDS and SMO-RDS using the same variable ordering is quite similar. Thus, it seems that it does not payoff the time spent to compute tighter lower bounds. However, both algorithms perform better when using the LEX-M variable ordering. Note that, as shown in Section B.1, the bandwidth given by LEX-M ordering is always smaller than the ones given by lexicographical ordering.

6.3.4 Scheduling of an EOS

Figure 6.11 reports the results on the scheduling of an EOS benchmark. The first column of the figure indicates the name of the instance. The second

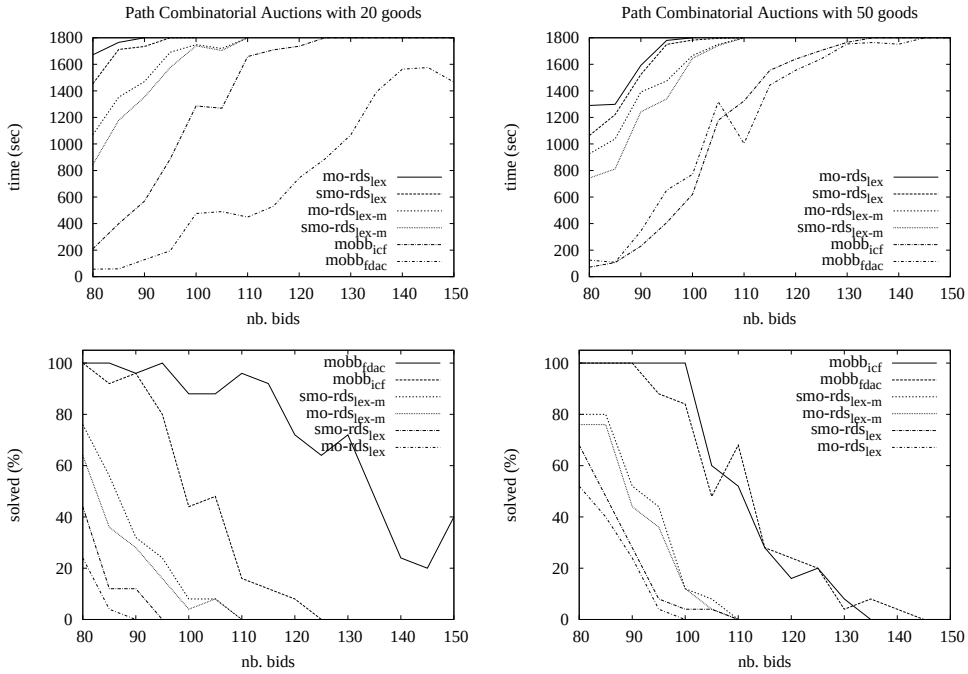

Figure 6.10: Experimental results on risk-conscious combinatorial auctions for 20 and 50 goods, respectively. Path distribution. Mean values on samples of size 25. Time limit 1800 seconds.

column tells the number of constraints. The third column indicates the graph bandwidth using the lexicographical and MWO order separated by an "/". The fourth and fifth columns show the cpu time in seconds for MO–RDS and SMO–RDS using the previous two variable orderings. The last column contains the cpu time of the best algorithm from the previous chapter (i.e., MO-BB$_{\texttt{icf}}$). Symbol "-" indicates that the algorithm cannot solve the subproblem within the time limit.

MO-RDS and SMO-RDS are clearly the best alternatives for this benchmark. These results indicates that the time spent in computing a lower bound frontier more sophisticated than $\texttt{lbf}_{\texttt{icf}}$ is worthy. This result is somehow expectable, since mono-objective RDS already proved its efficiency in the single

Instance	# vars	# constrs	Lex / MWO			
			bandwidth	mo-rds	smo-rds	mo-bb$_{\texttt{icf}}$
1504(0,183)*	184	1329	46 / 54	- / -	1114 / -	-
1504(184,206)*	23	112	8 / 8	0 / 0	0 / 0	0.36
1504(462,508)*	47	301	21 / 20	0.61 / 0.58	0 / 0.3	-
1506(0,150)	151	1440	77 / 78	- / -	- / -	-
1506(151,228)	78	1107	71 / 69	424 / 41.22	- / 7.8	-
1506(229,317)	89	1349	69 / 72	90 / -	62 / -	-
1506(679,761)*	83	1243	30 / 30	1 / 2.5	1 / 3.3	-
1405(762,854)*	93	2193	49 / 49	2.31 / 3.58	1.58 / 3.14	-
1407(0,147)	148	1442	79 / 80	- / -	1625 / -	-
1407(148,247)	100	1678	80 / 80	866 / 26.12	- / 17.12	-
1407(248,378)	131	3063	103 / 112	366 / -	680 / -	-
1407(379,409)*	31	220	12 / 12	0 / 0	0 / 0	0.18
1407(413,429)*	17	87	9 / 9	0 / 0	0 / 0	0.02
1407(447,469)*	23	129	10 / 10	0 / 0	0 / 0	0
1407(494,553)*	60	1333	46 / 51	267 / -	- / -	-
1407(580,700)	121	2299	86 / 86	- / -	1769 / -	-
1407(701,761)	61	445	28 / 30	7 / 44.5	3 / 98.35	-
1407(762,878)*	117	2708	49 / 49	27 / 5.04	96 / 4.8	-

Figure 6.11: Experimental results on subproblems of the Spot5 instances with capacity constraint. Time limit 1800 seconds.

orbit instances and our algorithms are a natural generalization.

The efficiency of MO-RDS and SMO-RDS depends on the structure of each subinstance. When the lower bound of a given subproblem P^i is similar to the lower bound of each subproblem $P^i(x_i = a)$, MO-RDS outperforms SMO-RDS because the latter cannot take advantage of the specialized lower bound. However, when this is not the case, SMO-RDS computes a tighter lower bound and, as a result, SMO-RDS is more efficient than MO-RDS.

Finally, observe that for those instances where the two orderings lead to different bandwidth, MO-RDS and SMO-RDS are more efficient using the variable ordering leading to the smallest graph bandwidth. When both orderings lead to the same graph bandwidth, it seems that (S)MO-RDS using MWO outperforms the same algorithm using the lexicographical ordering (see 1407(148,247) and 1407(762,878) instances).

Since SMO-RDS has been proved to be an efficient alternative to solve each connected subpart of multiple-orbit instances, we focus on the resolution

Instance	constr.	SMO-RDS (sec.)
1504(0,183)*	1329	1114
1504(184,206)*	112	0
1504(356,461)*	840	13418
1504(462,508)*	301	0

Figure 6.12: Experimental results on instance 1504.

of instance 1504. This instance is very challenging and remains unsolved since its capacity constraint is not trivially satisfied. We solve to optimality each connected part of the problem using SMO-RDS with no time restriction. Figure 6.12 reports the cpu time for the most difficult subinstances. The most difficult piece, $1504(356, 461)$, is solved in less than 4 hours. As a consequence, we can sum the efficient frontier of each subproblem and extract the solution of the original problem: 161301. This instance is solved for the first time.

6.3.5 More Biobjective Minimum Vertex Cover

From these experiments, it may seem that russian doll algorithms are only able to solve problems with relatively small bandwidth. As noted in Section 2.4, the bandwidth of a constraint graph under a given order is always greater than its induced width under the same order. As we will see in the next Chapter, *inference* algorithms can solve this kind of problems very efficiently. Under these circumstances, it may seem that RDS algorithms would not be very useful.

The next experiment shows that RDS can solve problems with high bandwidth and induced width (and as a consequence, they cannot be solved by inference algorithms). This experiment also indicates that the graph bandwidth is not a very accurate measure to predict how hard the problem is for RDS.

We test on instances from the class of problems $(90, 500, 5)$, with two extra parameters: the initial graph bandwidth, noted B, and a percentage of constraints, noted P. The instances are generated as follows. We randomly

generate $500(1-P)$ constraints whose scope variables are separated by a distance lower than or equal to B according to lexicographical ordering. Then, we randomly generate $500P$ constraints without any limitation. We experiment on samples of size 25 for $B \in \{9, 18, 27, 36\}$ and varying the percentage of constraints P from 0.1 to 0.5 in steps of 0.05.

Figure 6.13 reports the cpu time in seconds for SMO-RDS algorithm under the lexicographical order for $B = 9$ and $B = 18$ (top left and top right, respectively), and $B = 27$ and $B = 36$ (middle left and middle right, respectively). All these plots report also the bandwidth (noted b^*), the induced width (noted w^*), and an alternative measure of the bandwidth that considers the mean bandwidth of each node (noted b^*_m) under lexicographical ordering. The figure also reports the solved percentage for each value of B (bottom).

Observe that the graph bandwidth as well as the induced width of each parameter configuration is relatively high. In particular, the induced width renders the application of inference algorithms unfeasible (as we will see in the next chapter). Although the graph bandwidth is almost the number of variables for all parameter configurations, the behaviour of SMO-RDS depends on the values of both B and P. As expected, if we fix the value of B and increase the percentage P, the efficiency of SMO-RDS decreases. Moreover, if we fix P and increase the value of B, the performance of SMO-RDS also decreases. The link between B and P is also exemplified in the plot of the solved percentage. Observe that as the value of B increases, the value of P for which SMO-RDS is able to solve completely the sample of each parameter configuration decreases.

6.4 Conclusions

This chapter has two main contributions. The first one is algorithmic, since we extend RDS and SRDS from mono-objective to multi-objective optimization. The new algorithms define two new valid lower and upper bound fron-

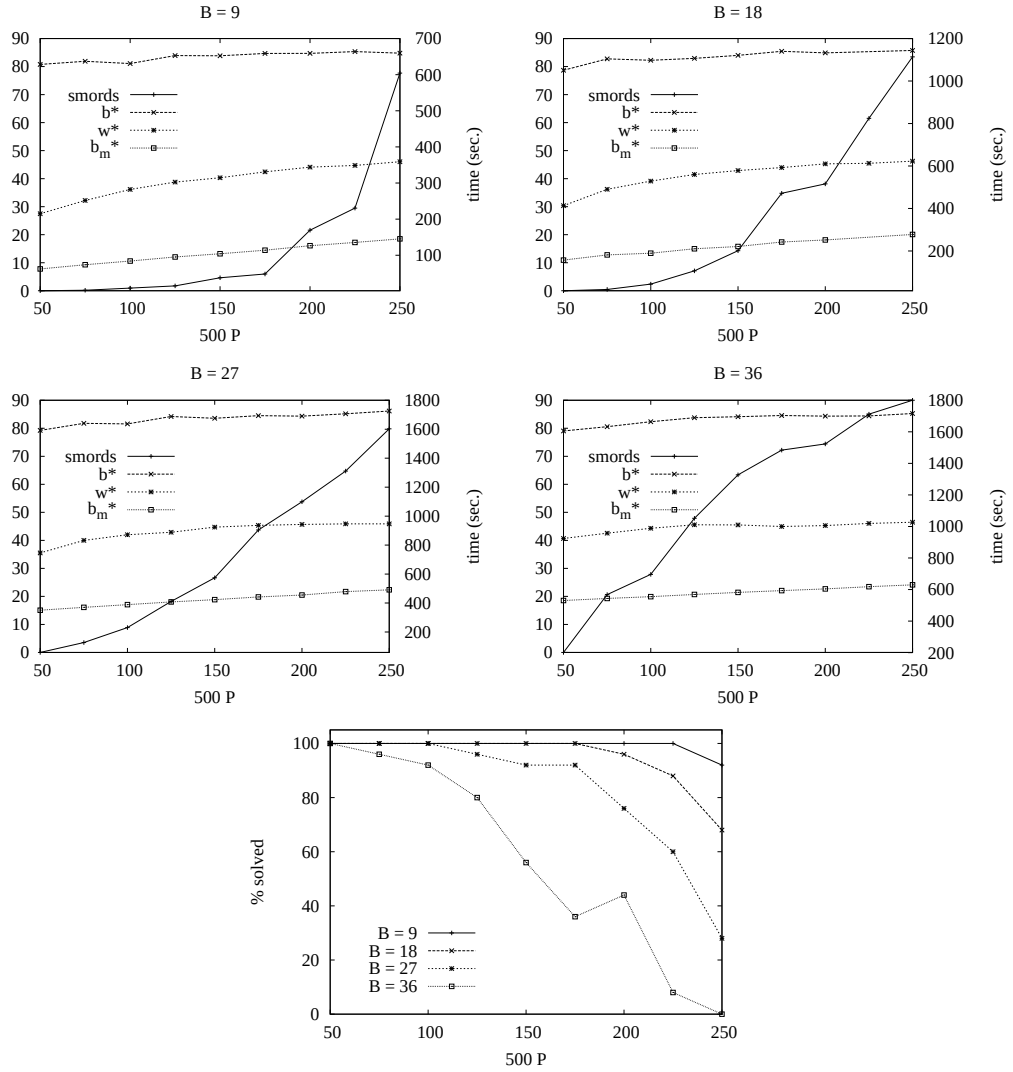

Figure 6.13: Influence of graph bandwidth on biobjective minimum weighted vertex cover instances with 90 nodes and 500 edges. Samples of size 25. Time limit 1800 seconds.

tiers, which improve the ones described in Section 5.2.1. The second contribution comes from our empirical work, where we show that our multi-objective

RDS is an efficient alternative in an important class of problems. Moreover, using multiple orbit Spot5 instances, we illustrate that sometimes it may be convenient to reformulate pure satisfaction or mono-objective optimization problems as multi-objective problems. This may be counterintuitive at first sight because multi-objective optimization is in general more difficult than mono-objective optimization. However, we show that the multi-objective perspective may bring to light desirable structural properties.

As suggested in the experimental results of the *biobjective minimum vertex cover* benchmark, the parameter able to characterize the efficiency of russian doll search algorithms should balance in an appropriate way the impact of nodes with high bandwidth with respect to nodes with relatively small bandwidth. Our proposal is to study some parameters as the mean (weighted or unweighted) among all nodes' bandwidth of the problem. Clearly, this matter deserves further research.

It is known that russian doll algorithms are very sensitive to heuristic decisions [103]. The current implementation of SMO-RDS is far from being fully optimized with respect to available mono-objective versions[2]. Thus, more efficient implementations may lead to improved results.

[2]ftp://ftp.cert.fr/pub/lemaitre/LVCSP

Chapter 7

Bucket Elimination

Decomposition methods are widely used in mono-objective optimization and *bucket elimination* is probably the most popular one. To the best of our knowledge, there is no previous research on decomposition method explicitly applied to multi-objective optimization problems. The purpose of this chapter is to address this lack by extending bucket elimination to multi-objective optimization. The new algorithm, called *multi-objective bucket elimination* (MO-BE), is a general inference algorithm able to solve multi-objective optimization problems with small induced width, as we theoretically and empirically demonstrate. Moreover, MO-BE is the starting point to describe its approximated version that computes a lower bound frontier of the given multi-objective problem (see Chapter 8).

The structure of the chapter is as follows. Section 7.1 describes bucket elimination. Then, Section 7.2 depicts a non-standard implementation of bucket elimination which will facilitate the comprehension of its extension to multi-objective optimization problems, introduced in Section 7.3. Section 7.4 reports some experimental results. Section 7.5 discusses related work and, finally, Section 7.6 gives some conclusions.

7.1 Bucket Elimination

Inference is a general problem solving technique for graphical models. This approach transforms a problem into an equivalent one through problem reformulation. The idea is to make explicit some knowledge that is implicit in the original problem. A *brute-force* inference can be described simply as marginalizing all variables out of the combination of all functions. Given a WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, this naive approach will explicitly compute the expression

$$\min_{\mathcal{X}} \{ \sum_{f \in \mathcal{F}} f \} \tag{7.1}$$

in two steps. First, it will compute the sum of all functions $F(\mathcal{X}) = \sum_{f \in \mathcal{F}} f$, and then it will compute the minimum entry $\min_{\mathcal{X}} \{F(\mathcal{X})\}$. This approach is time and space exponential in the number of variables, and thus impractical.

Bucket Elimination (BE) [34] (*non-serial dynamic programming* in [15] and *fusion algorithm* in [135]) solves mono-objective optimization problems by a sequence of problem transformations. These transformations are based in the sequential elimination of variables and taking advantage of the properties of the combination and marginalization operators and of the modularity of graphical models, as follows.

Given a static variable ordering, that we assume lexicographic without loss of generality, the previous expression can be rewritten as a recursion

$$F^i = \min_{x_i} \{ F^{i+1} \} \tag{7.2}$$

from $i = n$ down to 1 where, by definition, $F^{n+1} = \sum_{f \in \mathcal{F}} f$. By construction, given a tuple $t = (x_1 = a_1, \ldots, x_{i-1} = a_{i-1})$, $F^i(t)$ is the optimal cost of extending t to the eliminated variables $x_i, x_{i+1}, \ldots, x_n$. Then F^1, resulting from the elimination of the last variable x_1, is a zero-arity function (i.e., a constant) which is the optimal cost of P.

If F^i were computed naïvely, the space and time complexity of this recursion would be the same as for the naive approach. Instead, the following property, based on the commutativity and associativity of the + and *min*

operator (a.k.a. *distributivity of marginalization over combination* in [135]), can be exploited in each step of the recursion:

$$\min_{x_i}\{\sum_{f\in\mathcal{S}} f\} = \sum_{\substack{f\in\mathcal{S}\\ x_i\notin var(f)}} f + \min_{x_i}\{\sum_{\substack{f\in\mathcal{S}\\ x_i\in var(f)}} f\}$$

where $\mathcal{S}$ is an arbitrary set of functions. In words, when eliminating variable x_i, the only relevant functions are the ones containing x_i in their scope. This set of functions is the so-called *bucket* of x_i, noted $\mathcal{B}_i$. Then, the i^{th} step of the previous recursion can be further rewritten as,

$$F^i = \min_{x_i}\{\sum_{\substack{f\in\mathcal{F}^{i+1}\\ f\in\mathcal{B}_i}} f\} + \sum_{\substack{f\in\mathcal{F}^{i+1}\\ f\notin\mathcal{B}_i}} f \tag{7.3}$$

The elimination of variable x_i implies the replacement of the sum of the set of functions in $\mathcal{B}_i$ by a new function

$$g_i = \min_{x_i}\{\sum_{\substack{f\in\mathcal{F}^{i+1}\\ f\in\mathcal{B}_i}} f\}$$

which does not mention x_i. g_i is the only function that must be computed explicitly. It is easy to see that the space and time complexity of computing this function is exponential in its arity $|var(g_i)| = |\bigcup_{f\in\mathcal{B}_i} var(f)|$. As a consequence, the space and time complexity of the previous recursion is exponential in the largest arity function computed.

It is also important to note that g_i results from the elimination of variables $x_i, x_{i+1}, \ldots, x_n$ from a subset of original functions. Then, by construction, tuple $t = (x_1 = a_1, \ldots, x_{i-1} = a_{i-1})$ can be consistently extended to the eliminated variables $x_i, x_{i+1}, \ldots, x_n$ from those original functions with cost $g_i(t)$.

Finally, the modularity of graphical models allows us to describe the previous recursion as a sequence of problem transformations. Each function F^i is the objective function of a problem whose set of functions $\mathcal{F}^i$ is composed

function BE$(\mathcal{X}, \mathcal{D}, \mathcal{F})$
1. **for each** $i = n \dots 1$ **do**
2. $\quad \mathcal{B}_i := \{f \in F |\ x_i \in var(f)\}$
3. $\quad g_i := \min_{x_i}\{\sum_{f \in \mathcal{B}_i} f\}$;
4. $\quad F := (F \cup \{g_i\}) - \mathcal{B}_i$;
5. **endfor**
6. $t := \lambda$;
7. **for each** $i = 1 \dots n$ **do**
8. $\quad v := \text{argmin}_{a \in D_i}\{(\sum_{f \in \mathcal{B}_i} f)(t \cdot (x_i = a))\}$
9. $\quad t := t \cdot (x_i = v)$;
10. **endfor**
11. **return** (g_1, t);

endfunction

Figure 7.1: Bucket Elimination. Given a WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, the algorithm returns a constant function g_1 (i.e, $var(g_1) = \emptyset$) with the optimal cost of P, along with one optimal assignment t.

by each summation of F^i. Then, the execution of BE can be viewed as a recursion

$$\mathcal{F}^i = \{\min_{x_i}\{\sum_{\substack{f \in \mathcal{F}^{i+1} \\ f \in \mathcal{B}_i}} f\}\} \ \cup \ \{f \in \mathcal{F}^{i+1} \mid f \notin \mathcal{B}_i\} \tag{7.4}$$

from $i = n$ down to 1 where, by definition, $\mathcal{F}^{n+1}$ is the set of original functions $\mathcal{F}$. In each step, BE transforms the set of functions of the previous problem by replacing the set of functions in $\mathcal{B}_i$ by a new function g_i that summarizes their effect. The elimination of the last variable computes the set $\mathcal{F}^1$, which contains one zero-arity function g_1. As we have seen, $g_1()$ is the optimal cost of P.

Figure 7.1 shows an algorithmic description of BE for solving WCSP

problems. The input of the algorithm is the WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ to be solved. The output is the optimum cost of P ($\texttt{opt}(P)$) along with one optimal assignment t (i.e., $F(t) = \texttt{opt}(P)$).

BE works in two phases. In the first phase (lines 1-5), the algorithm implements the recursion in expression 7.4. That is, it eliminates variables one by one, from last to first, according to a given order o, that we assume lexicographic. The elimination of variable x_i is done as follows: the algorithm computes bucket $\mathcal{B}_i$ (line 2). Next, it computes a new function g_i by summing all functions in $\mathcal{B}_i$ and subsequently eliminating x_i (line 3). Then, the set of current functions $\mathcal{F}$ is updated by removing the functions in $\mathcal{B}_i$ and adding g_i (line 4). The new $\mathcal{F}$ does not contain x_i (all functions mentioning x_i were removed and g_i, by construction, does not contain x_i) but preserves the value of the optimal cost. The elimination of the last variable produces an empty-scope function g_1 (i.e., a constant) which is the optimal cost of the problem.

The second phase (lines 6-11) generates an optimal assignment of variables. It uses the set of buckets that were computed in the first phase. Starting from an empty assignment t (line 6), variables are assigned from first to last according to o. The optimal domain value for x_i is the best one regarding the extension of t with respect to the sum of functions in $\mathcal{B}_i$ (lines 8,9). We use argmin to denote the argument producing the minimum valuation. Let the *g_i-subproblem* denote the subproblem formed by all the original cost functions involved in the computation of g_i. Let t be an assignment of variables $x_1, \ldots, x_{i-1}$. The correctness of BE is a direct consequence of the fact that when processing bucket $\mathcal{B}_i$, $g_i(t[var(g_i)])$ is the cost of the best extension of t to variables $x_i, x_{i+1}, \ldots, x_n$ in the g_i-subproblem. Note that the g_1-subproblem is the original problem P.

Example 7.1.1 *Consider a WCSP instance with seven variables and the following set of cost functions,*

$$\mathcal{F} = \{f_1(x_6, x_5, x_4), f_2(x_6, x_5, x_3), f_3(x_5, x_3, x_2), f_4(x_6, x_4, x_2),$$

$$f_5(x_7, x_2, x_1), f_6(x_7, x_6, x_1)\}$$

The execution of BE along the lexicographical variable ordering leads to the following trace,

Bucket	
$\mathcal{B}_7$*:*	$f_6(x_7, x_6, x_1),\ f_5(x_7, x_2, x_1)$
$\mathcal{B}_6$*:*	$g_7(x_6, x_2, x_1) = \min_{x_7}\{f_6 + f_5\}$, $f_4(x_6, x_4, x_2),\ f_2(x_6, x_5, x_3),\ f_1(x_6, x_5, x_4)$
$\mathcal{B}_5$*:*	$g_6(x_5, x_4, x_3, x_2, x_1) = \min_{x_6}\{f_1 + f_2 + f_4 + g_7\},\ f_3(x_5, x_3, x_2)$
$\mathcal{B}_4$*:*	$g_5(x_4, x_3, x_2, x_1) = \min_{x_5}\{f_3 + g_6\}$
$\mathcal{B}_3$*:*	$g_4(x_3, x_2, x_1) = \min_{x_4}\{g_5\}$
$\mathcal{B}_2$*:*	$g_3(x_2, x_1) = \min_{x_3}\{g_4\}$
$\mathcal{B}_1$*:*	$g_2(x_1) = \min_{x_2}\{g_3\}$
Result:	$g_1() = \min_{x_1}\{g_2\}$

The first column indicates the bucket $\mathcal{B}_i$ being treated, and the second column shows the functions included in that bucket. Since the algorithm considers the lexicographical ordering of the variables, buckets are processed from $\mathcal{B}_7$ down to $\mathcal{B}_1$. The first bucket processed is $\mathcal{B}_7$. It contains functions f_6 and f_7 because they are the functions in the problem having x_7 in their scope. The combination of these two functions and the subsequent elimination of variable x_7 leads to function g_7. Since the highest variable in the scope of g_7 according to the variable ordering is x_6, it is place in the bucket of this variable $\mathcal{B}_6$. Bucket $\mathcal{B}_6$ also contains the original functions f_4, f_2 and f_1, because x_6 is the highest variable in their scope. Then, a new function g_6 is computed summing all the functions in $\mathcal{B}_6$ and eliminating variable x_6. g_6 is placed in bucket $\mathcal{B}_5$ for the same reason as before. When processing the last bucket $\mathcal{B}_1$, the result is a zero-arity function g_1, which is the optimal cost of the problem.

In general, the result of combining functions or eliminating variables cannot be expressed intensionally by algebraic expressions. Therefore, we assume

functions to be extensionally stored in tables. Thus, the space complexity of storing function f is $O(d^{|var(f)|})$.

Theorem 7.1.1 *[34] The complexity of* BE *along ordering o is time $O(e \times d^{w^*(o)+1})$ and space $O(n \times d^{w^*(o)})$, where e is the number of functions, d is the largest domain size, n is the number of variables and $w^*(o)$ is the induced width under the corresponding variable ordering (see Definition 2.4.4).*

A clear consequence of Theorem 7.1.1 is that BE is a suitable algorithm for WCSP problems with small induced width. Otherwise, the algorithm suffers from large storage demands, which renders BE unfeasible with current technology.

7.2 A non-standard implementation of Bucket Elimination

In this section we provide a non-standard implementation of the second phase of the BE algorithm (Figure 7.1, lines 6-10). Actually, it is a slight modification of the approach used in [135] for retrieving an optimal solution. Although it may look unnecessarily complex for BE, it will facilitate the comprehension of the new algorithm MO-BE introduced in the next section. The idea is to retrieve the optimal solution by keeping track of the optimal cost of the different subproblems contained in each bucket.

Let $\mathcal{B}_i = \{f_{i_1}, \ldots, f_{i_{m_i}}\}$ be the set of cost functions of bucket $\mathcal{B}_i$. Each cost function f_{i_k} is either an original function or the result of processing a higher bucket $\mathcal{B}_j$ (i.e., $f_{i_k} = g_j$). We define $db(f_{i_k})$ as the *departure bucket* for function f_{i_k}, that is, the bucket where the function was generated. Therefore, $db(f_{i_k}) = i$ if f_{i_k} is an original function, and $db(f_{i_k}) = j$ if $f_{i_k} = g_j$. For instance, in the previous example the departure bucket of f_6 is $db(f_6) = 7$, because f_6 is an original function of bucket $\mathcal{B}_7$. Similarly, $db(f_3) = 5$. The departure bucket of g_7 is $db(g_7) = 7$, because g_7 is the result of computing the functions in $\mathcal{B}_7$. Similarly, $db(g_5) = 5$.

6. $t := \lambda$;
7. $C[1] := g_1$;
8. **for each** $i = 1 \dots n$ **do**
9. let $\mathcal{B}_i = \{f_{i_1}, f_{i_2}, \dots, f_{i_{m_i}}\}$
10. $b := pop(\{a \in D_i \mid (\sum_{k=1}^{m_i} f_{i_k})(t \cdot (x_i = a)) = C[i]\})$;
11. $t := t \cdot (x_i = b)$;
12. $(v_1, \dots, v_{m_i}) := (f_{i_1}(t), \dots, f_{i_{m_i}}(t))$;
13. **for each** $k = 1 \dots m_i$ **do if** $db(f_{i_k}) \neq i$ **then** $C[db(f_{i_k})] := f_{i_k}(t)$;
14. **return**(g_1, t);

Figure 7.2: Second phase of the Bucket Elimination with a non-standard implementation.

As in standard BE, the new second phase of the algorithm (Figure 7.2) generates in t an optimal assignment of variables, considering them one at the time, from first to last. We use an array $C[1 \dots n]$. Each $C[i]$ will store the cost contribution of g_i to the optimal solution (namely, the contribution of the g_i-subproblem). Initially, t is an empty assignment λ (line 6). Clearly, $C[1]$ is set to g_1 (line 7). The optimal value for x_1 is any domain value $b \in \mathcal{D}_1$ such that $C[1] = \sum_{k=1}^{m_i} f_{1_k}(t \cdot (x_1 = b))$. In line 10 one such value is selected and in line 11 added to t (i.e., $t = t \cdot (x_1 = b)$). The contribution of each function $f_{1_k} \in \mathcal{B}_1$ to the cost $C[1]$ is $f_{1_k}(t)$. Therefore, each contribution $f_{1_k}(t)$ is propagated to the C entry of the corresponding departure bucket $C[db(f_{1_k})]$ (lines 12-13). The same procedure is repeated for each variable x_i in increasing order.

7.3 Multi-objective Bucket Elimination

Multi-objective Bucket Elimination (MO-BE) [120] extends bucket elimination to the multiobjective context. Given a MO-WCSP problem $P =$

$(\mathcal{X}, \mathcal{D}, \mathcal{F})$, MO-BE computes the efficient frontier of P (i.e., $\mathcal{E}(P)$) by a sequence of problem reductions where, at each step, the algorithm eliminates one new variable.

The execution of MO-BE can be described as a recursion

$$\mathcal{F}^i = \{ \min_{x_i}^{mo} \{ \sum_{\substack{f \in \mathcal{F}^{i+1} \\ f \in \mathcal{B}_i}}^{mo} f \} \} \cup \{ f \in \mathcal{F}^{i+1} \mid f \notin \mathcal{B}_i \} \tag{7.5}$$

from $i = n$ down to 1 where, by definition, $\mathcal{F}^{n+1} = \mathcal{F}$. It is worth noting that $\mathcal{F}^i$ is a set of frontier functions. Let $t = (x_1 = a_1, \ldots, x_{i-1} = a_{i-1})$ be a tuple and let g_i be the new frontier function computed during the elimination of x_i. It is important to note that g_i involves the elimination of variables $x_i, \ldots, x_n$ from a subset of original functions. By construction, $g_i(t)$ is a frontier such that each cost vector $\vec{v} \in g_i(t)$ is a non-dominated extension of t to the eliminated variables $x_i, \ldots, x_n$ in those original functions. In other words, tuple t can be consistently extended to variables $x_i, \ldots, x_n$ with cost v_j for the j^{th} objective function, where v_j is the j^{th} component of $\vec{v}$. The efficient frontier $\mathcal{E}(P)$ is the function computed when the last variable is eliminated (i.e. $F^1() = \mathcal{E}(P)$).

Figure 7.3 shows MO-BE, the generalization of BE to MO-WCSP. Its structure is similar to standard BE. In the following, we discuss the main differences. MO-BE receives a MO-WCSP instance $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$. The first phase of the algorithm (lines 1-5) computes $\mathcal{E}(P)$ by implementing the recursion 7.5. It works as BE, the only difference being that frontier functions are used instead of standard cost functions. After the first phase, $g_1()$ contains a set of points in the space of solutions, which is exactly the efficient frontier $\mathcal{E}(P)$ of the problem.

Let $g_1()$ contain r vector points $\{\vec{v_1}, \vec{v_2}, \ldots, \vec{v_r}\}$. The second phase (lines 7-20) computes one efficient solution t_j for every element $\vec{v_j} \in g_1()$. The idea is to retrieve the efficient solution keeping track of the cost contribution of each g_i to the solution. In this case, the array $C[i]$ will store a non-dominated cost vector attainable from g_i. Initially, $t_j = \lambda$ and $C[1] = \vec{v_j}$

function MO-BE$(\mathcal{X}, \mathcal{D}, \mathcal{F})$
1. **for each** $i = n \dots 1$ **do**
2. $\quad \mathcal{B}_i := \{f \in \mathcal{F} \mid x_i \in var(f)\}$;
3. $\quad g_i := \min_{x_i}^{mo} \{\sum_{f \in \mathcal{B}_i} f\}$;
4. $\quad \mathcal{F} := (\mathcal{F} \cup \{g_i\}) - \mathcal{B}_i$;
5. **endfor**
6. let $g_1() = \{\vec{v_1}, \vec{v_2}, \dots, \vec{v_r}\}$;
7. **for each** $j = 1 \dots r$ **do**
8. $\quad t_j := \lambda$;
9. $\quad C[1] := \vec{v_j}$;
10. $\quad$ **for each** $i = 1 \dots n$ **do**
11. $\quad\quad$ let $\mathcal{B}_i = \{f_{i_1}, \dots, f_{i_{m_i}}\}$;
12. $\quad\quad$ **for each** $a \in D_i$ **do**
13 $\quad\quad\quad S_a = \{(\vec{v_1}, \dots, \vec{v_{m_i}}) \mid \sum_{k=1}^{m_i} \vec{v_k} = C[i],\ \forall k,\ \vec{v_k} \in f_{i_k}(t \cdot (x_i = a))\}$;
14. $\quad\quad$ **endfor**
15. $\quad\quad b := \text{pop}(\{a \in D_i \mid S_a \neq \emptyset\})$;
16. $\quad\quad t_j := t_j \cdot (x_i = b)$;
17. $\quad\quad (\vec{v_1}, \dots, \vec{v_{m_i}}) := \text{pop}(S_b)$;
18. $\quad\quad$ **for each** $k = 1 \dots m_i$ **do if** $db(f_{i_k}) \neq i$ **then** $C[db(f_{i_k})] := \vec{v_k}$;
19. $\quad$ **endfor**
20. **endfor**
21. **return** $(g_1 = \{\vec{v_1}, \vec{v_2}, \dots, \vec{v_r}\}, \{t_1, t_2, \dots, t_r\})$;

endfunction

Figure 7.3: Description of MO-BE. The input is a MO-WCSP instance $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$. The output is g_1, a zero-arity frontier function which contains the efficient frontier of P and, for each element $\vec{v_k} \in g_1$, an efficient solution t_k.

is the cost vector for which the efficient solution is searched. For each j, variables are considered in increasing order $x_1, \ldots, x_n$ (line 10). The optimal domain value $a \in D_1$ for x_1 is any one such that $C[1] \in \sum_{k=1}^{m_1} f_{1_k}(t \cdot (x_1 = a))$. Since each $f_{1_k}(t \cdot (x_1 = a))$ contains a set of non-dominated cost vectors, there must exist at least one combination of cost vectors $(\vec{v_1}, \ldots, \vec{v_{m_1}})$ where each $\vec{v_k} \in f_{1_k}(t \cdot (x_1 = a))$, such that $C[1] = \sum_{k=1}^{m_1} \vec{v_k}$. Let S_a be the set of such combinations for domain value a (lines 12–14). Tuple t_j is extended to variable x_1 with a domain value b for which exists at least one combination (lines 15-16). One arbitrary combination $(\vec{v_1}, \ldots, \vec{v_{m_1}}) \in S_b$ is selected in line 17. The contribution to the solution of each $f_{1_k} \in \mathcal{B}_1$ is $\vec{v_k}$. Therefore, it is possible to update the array C of each departure bucket (line 18). The same procedure is repeated for each variable. At the end of the process, t_j is an efficient solution with cost vector $\vec{v_j}$.

Property 7.3.1 *In a problem with one objective function (i.e., $p = 1$), the algorithm MO-BE is equivalent to BE.*

Example 7.3.1 *Consider a MO-WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ where:*

- $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$
- $\mathcal{D} = \{\{D_i\}_{i=1}^4\}$ *where* $D_i = \{0, 1\}$
- $\mathcal{F} = \{h_1, h_2, h_3, \{p_i\}_{i=1}^4, p_{23}, \{w_i\}_{i=1}^4, \{v_i\}_{i=1}^4\}$ *where*

$$h_1(x_1, x_3) = \begin{cases} \{(0,0,0)\} & (x_1 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \\ \top & otherwise \end{cases}$$

$$h_2(x_3, x_4) = \begin{cases} \{(0,0,0)\} & x_3 \vee x_4 \\ \top & otherwise \end{cases}$$

$$h_3(x_2, x_4) = \begin{cases} \{(0,0,0)\} & \overline{x_2 \wedge x_4} \\ \top & otherwise \end{cases}$$

$$p_i(x_i) = \begin{cases} \{(0,0,0)\} & x_i = 1 \\ \{(i,0,0)\} & x_i = 0 \end{cases}$$

$$p_{23}(x_2, x_3) = \begin{cases} \{(0,0,0)\} & x_2 \wedge x_3 \\ \{(3,0,0)\} & otherwise \end{cases}$$

$$w_i(x_i) = \begin{cases} \{(0,5-i,0)\} & x_i = 1 \\ \{(0,0,0)\} & x_i = 0 \end{cases}$$

$$v_i(x_i) = \begin{cases} \{(0,0,i)\} & x_i = 1 \\ \{(0,0,0)\} & x_i = 0 \end{cases}$$

and the top value is $\top = \{(\infty, \infty, \infty)\}$.

The trace of the algorithm under lexicographical ordering is:

- *Input: the MO-WCSP problem* P.
- *Elimination of* x_4: $\mathcal{B}_4 = \{h_2, h_3, p_4, w_4, v_4\}$. *Their sum is* $b_4(x_2, x_3, x_4)$,

$$b_4(001) = \{(0,1,4)\} \quad b_4(010) = \{(4,0,0)\}$$

$$b_4(011) = \{(0,1,4)\} \quad b_4(110) = \{(4,0,0)\}$$

Note that $b_4(000) = b_4(100) = b_4(101) = b_4(111) = \top$. *As a consequence, those assignments cannot lead to any problem solution. In*

the sequel, we only indicate assignments which may lead to a problem solution.

Projecting x_4 *out of* b_4 *produces* $g_4(x_2, x_3)$,

$$g_4(00) = \{(0,1,4)\} \quad g_4(01) = \{(4,0,0),(0,1,4)\} \quad g_4(11) = \{(4,0,0)\}$$

- *Elimination of* x_3*:* $\mathcal{B}_3 = \{g_4, h_1, p_3, p_{23}, w_3, v_3\}$*. Their sum is* $b_3(x_1, x_2, x_3)$,

$$b_3(001) = \{(7,2,3),(3,3,7)\} \quad b_3(011) = \{(4,2,3)\} \quad b_3(100) = \{(6,1,4)\}$$

Projecting x_3 *out of* b_3 *produces* $g_3(x_1, x_2)$,

$$g_3(00) = \{(7,2,3),(3,3,7)\} \quad g_3(01) = \{(4,2,3)\} \quad g_3(10) = \{(6,1,4)\}$$

- *Elimination of* x_2*:* $\mathcal{B}_2 = \{g_3, p_2, w_2, v_2\}$*. Their sum is* $b_2(x_1, x_2)$,

$$b_2(00) = \{(9,2,3),(5,3,7)\} \quad b_2(01) = \{(4,5,5)\} \quad b_2(10) = \{(8,1,4)\}$$

Projecting x_2 *out of* b_2 *produces* $g_2(x_1)$,

$$g_2(0) = \{(9,2,3),(5,3,7),(4,5,5)\} \quad g_2(1) = \{(8,1,4)\}$$

- *Elimination of* x_1*:* $\mathcal{B}_1 = \{g_2, p_1, w_1, v_1\}$*. Their sum is* $b_1(x_1)$,

$$b_1(0) = \{(10,2,3),(6,3,7),(5,5,5)\} \quad b_1(1) = \{(8,5,5)\}$$

Projecting x_1 *out of* b_1 *produces* $g_1 = \{(10,2,3),(6,3,7),(5,5,5)\}$

Note that $(8,5,5)$ *is not an efficient cost vector as it is dominated by* $(5,5,5)$.

Therefore, the problem has three Pareto optimal solutions. We show how to retrieve the one with costs $(10,2,3)$*:*

- *Initially,* $t = \lambda$*, and* $C[1] = (10,2,3)$.

- *Variable x_1 assignment: there are two values for x_1,*

$$t = (x_1 = 0), \qquad S_0 = \{((9,2,3),(1,0,0),(0,0,0),(0,0,0))\}$$
$$t = (x_1 = 1), \qquad S_1 = \{\}$$

Only value 0 satisfies the sum of frontier functions in $\mathcal{B}_1$ because S_0 is not empty. Therefore, t is updated to $(x_1 = 0)$ and the cost contribution of the departure bucket of every non original frontier function in $\mathcal{B}_1$ is updated with its corresponding $\vec{v_j} \in (\vec{v_1}, \ldots, \vec{v_4})$. In this case, there is only one non original function, g_2. Therefore, $C[db(g_2)] = C[2] = (9,2,3)$.

- *Variable x_2 assignment: there are two values for x_2,*

$$t = (x_1 = 0, x_2 = 0), \qquad S_0 = \{((7,2,3),(2,0,0),(0,0,0),(0,0,0))\}$$
$$t = (x_1 = 0, x_2 = 1), \qquad S_1 = \{\}$$

Only value 0 satisfies the sum of frontier functions in $\mathcal{B}_2$ because S_0 is not empty. Therefore, t is updated to $(x_1 = 0, x_2 = 0)$ and $C[db(g_3)] = C[3] = (7,2,3)$.

- *Variable x_3 assignment: there are two values for x_3,*

$$t = (x_1 = 0, x_2 = 0, x_3 = 0), \qquad S_0 = \{\}$$
$$t = (x_1 = 0, x_2 = 0, x_3 = 1), \qquad S_1 = \{((4,0,0),(0,0,0),(0,0,0),(3,0,0),(0,2,0),(0,0,3))\}$$

Only value 1 satisfies the sum of frontier functions in $\mathcal{B}_3$ as S_1 is not empty. Therefore, t is updated to $(x_1 = 0, x_2 = 0, x_3 = 1)$ and $C[db(g_4)] = C[4] = (4,0,0)$.

- *Variable x_4 assignment: there are two values for x_4,*

$$t = (x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0), \qquad S_0 = \{((0,0,0),(0,0,0),(4,0,0),(0,0,0),(0,0,0))\}$$
$$t = (x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1), \qquad S_1 = \{\}$$

Only value 0 *satisfies the sum of frontier functions in* $\mathcal{B}_4$ *because* S_0 *is not empty. Therefore,* t *is updated to* $(x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0)$. *As there is no original function, the cost contribution vector* C *is not updated.*

As a result, the Pareto optimal solution with cost vector $(10, 2, 3)$ *is* $(x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0)$.

The result of combining and eliminating frontier functions is assumed to be extensionally stored in tables. Given a frontier function f, it has $O(d^{|var(f)|})$ entries. Each entry of f stores a frontier. Thus, the space complexity of storing function f is

$$O(d^{|var(f)|} \times \max_{\{t|var(t)\subseteq\mathcal{X}\}}\{|f(t)|\})$$

In general, the maximum number of non-dominated cost vectors in a frontier is bounded, as shown in the following observation.

Observation 7.3.1 *Let* P *be a MO-WCSP problem with* p *objective functions. The maximum possible number of non-dominated cost vectors among all frontiers in* P *is given by the range of possible costs of each objective function. Without loss of generality, we can assume that for the* j^{th} *objective function this range is bounded by a maximum cost* K_j. *In the worst case,* K_j *is*

$$O(\sum_{f\in\mathcal{F}} \max_{\substack{\{t|var(t)\subseteq\mathcal{X}\}\\ f(t)=\{(v_1,\ldots,v_j,\ldots,v_p)\}}} \{v_j\})$$

Then, the maximum possible size of any frontier *is*

$$O(\prod_{j=1}^{p-1} K_j)$$

Observe that K_p *does not appear in the maximum possible size of any frontier. Since the order of the different objective functions is arbitrary, a straightforward optimization consists on leaving the largest* K_j *for the last position.*

Moreover, it is worth noting that this maximum possible number of non-dominated cost vectors is a very pessimistic worst-case estimate for frontiers size. In practice, this number is usually much smaller.

Theorem 7.3.1 *MO-BE is space* $O(n \times \prod_{j=1}^{p-1} K_j \times d^{w^*})$ *and time* $O(e \times \prod_{j=1}^{p-1} K_j^2 \times d^{w^*+1})$*, where* n *is the number of variables,* e *is the number of cost functions,* w^* *is the graph induced width,* K_j *is the maximum possible cost of the* j^{th} *objective function,* p *is the number of objective functions, and* d *is the largest domain size.*

Proof Let f be an arbitrary frontier function of arity r. Let K_j be the maximum possible cost of objective function j. In the worst case K_j is $O(\sum_{f \in \mathcal{F}_j} \max_{t, var(t) \subseteq \mathcal{X}} \{f(t)\})$ (see Observation 7.3.1). Observe that the space complexity of f is $O(\prod_{j=1}^{p-1} K_j \times d^r)$ because: there are $O(d^r)$ different instantiations of the problem variables and, for each instantiation, there may be up to $O(\prod_{j=1}^{p-1} K_j)$ undominated instantiations. Since the largest arity among the functions that MO-BE needs to store is bounded by $O(w^*)$ and there are e such functions, the space and time complexities clearly hold. ∎

7.4 Experimental Results

The purpose of these experiments is to analyze the suitability of MO-BE. In Section 7.3, we showed that the applicability of MO-BE depends on the problem's induced width. Therefore, we selected instances with induced width that we could handle with our computer (i.e., below 24). Variable orderings with small induced width were found with the min-degree heuristic [35].

We compare the performance of MO-BE versus the best approach from the previous chapters. Namely, the best among the following:

- MO-BB$_{\texttt{icf}}$.
- ϵ-constraint.
- MO-BB$_{fdac}$.

- MO-RDS and MO-SRDS.

We test MO-BE in the *Max-SAT-ONE*, *biobjective minimum vertex cover* and the *scheduling of an EOS* benchmarks (for a detailed description of the benchmarks see Appendices B.3, B.4, and B.2, respectively). We disregard the *risk-conscious combinatorial auctions* benchmark because as shown in Section B.1, the induced width of all instances surpass our limit.

The time limit for all the experiments is 1800 seconds.

7.4.1 Max-SAT-ONE

We found 36 Max-SAT-ONE instances with induced width below 24 which was our limit. Figure 7.4 reports the obtained results. The first, second and third columns contain the name, the efficient frontier size and the induced width of each instance, respectively. The last four columns indicate the cpu time in seconds required by MO-BE, MO-BB$_{\texttt{icf}}$, ϵ-constraint and MO-BB$_{fdac}$, respectively. A "-" indicates that the algorithm does not terminate in 1800 seconds.

It can be observed that MO-BE is the best approach for all instances. In accordance with the complexity analysis, the performance of MO-BE grows exponentially with the induced width. All the instances with small induced width (*dubois* and *pret*) are solved instantly. The *aim* and *ssa* instances, which have larger induced width, are still solved in less than half an hour. In particular, MO-BE is the only algorithm able to solve *ssa* instances within the time limit. The only exceptions are instances *ssa2670-130* and *ssa2670-141*.

This experiment confirms once more that with current computers, it is the space and not the time what limits the applicability of decomposition methods.

Instance	$\|\mathcal{E}\|$	w*	MO-BE	mo–bb$_{\texttt{icf}}$	ϵ–constraint	mo–bb$_{fdac}$
dubois20	1	3	0	125.98	0.002	34.87
dubois21	1	3	0	260.11	0.003	72.01
dubois22	1	3	0	543.91	0.003	149.16
dubois23	1	3	0	1131.71	0.001	308.26
dubois24	1	3	0	-	0.002	637.2
dubois25	1	3	0	-	0.003	1313.52
dubois26	1	3	0	-	-	-
dubois27	1	3	0	-	-	-
dubois28	1	3	0	-	-	-
dubois29	1	3	0	-	-	-
dubois30	1	3	0	-	-	-
dubois50	1	3	0	-	-	-
dubois100	1	3	0	-	-	-
pret60_25	20	4	0	-	-	-
pret60_40	14	4	0	-	-	-
pret60_60	6	4	0	630.34	-	1646.27
pret60_75	1	4	0	165.03	0.002	99.03
pret150_25	50	4	0	-	-	-
pret150_40	35	4	0	-	-	-
pret150_60	15	4	0	-	-	-
pret150_75	1	4	0	-	-	-
aim-50-1_6-no-1	8	15	1.97	48.91	1663.47	39.83
aim-50-1_6-no-2	10	20	26.97	30.62	1202.1	197.69
aim-50-1_6-no-3	10	19	7.31	46.95	-	60.71
aim-50-1_6-no-4	10	20	51.24	18.44	-	112.93
aim-50-1_6-yes1-1	10	18	13.04	20.89	-	51.55
aim-50-1_6-yes1-2	8	16	3.47	24.93	-	12.28
aim-50-1_6-yes1-3	10	19	26.04	24.6	-	25.62
aim-50-1_6-yes1-4	8	19	23.68	5.74	780.23	9
aim-50-2_0-no-4	10	23	444.54	25.2	-	70.05
aim-50-2_0-yes1-3	14	23	487.88	72.11	-	443.72
aim-50-2_0-yes1-4	14	21	20.59	150.78	-	1000.52
ssa0432-003	140	18	895.45	-	-	-
ssa2670-130	> 0	24	-	-	-	-
ssa2670-141	> 0	21	-	-	-	-
ssa7552-158	323	11	17	-	-	-
ssa7552-159	323	11	6.99	-	-	-
ssa7552-160	328	14	98.11	-	-	-

Figure 7.4: Experimental results on Max-SAT-ONE problems with small induced width. Time limit 1800 seconds.

7.4.2 Biobjective Weighted Minimum Vertex Cover

We test on instances from the following four classes of biobjective weighted minimum vertex cover problems: $(60, 100, 5)$, $(70, 100, 5)$, $(80, 100, 5)$ and

N (nb. vars)	E (nb. edges)	w*	MO-BE		MO–BB$_{fdac}$	
			Time (sec.)	Solved (%)	Time (sec.)	Solved (%)
60	100	9.5	0,1944	100%	0,45	100%
70		8.5	0,0792	100%	2,42	100%
80		7.3	0,0284	100%	9,69	100%
90		6.1	0,0096	100%	32,70	100%

Figure 7.5: Experimental results on biobjective weighted minimum vertex cover problems. Parameter C is set to 5. Mean values on 25 instances for each parameter configuration. Time limit 1800 seconds.

$(90, 100, 5)$. The other classes of problems described in Appendix B.4 have an induced width that surpasses our limit.

Figure 7.5 reports the obtained results. The first, second and third columns indicate the number of variables, the number of edges and the mean induced width of each parameter configuration, respectively. The fourth and fifth columns report the mean time in seconds required by MO-BB to solve each parameter configuration and the percentage of solved instances. The sixth and seventh columns report the same information for MO-BB$_{fdac}$, which was the best approach in this benchmark so far.

Again, these results confirm that MO-BE is very efficient in instances with small induced width. Observe that, although all instances are solved almost instantly, the performance of MO-BE increases as the induced width decreases.

7.4.3 Scheduling of an EOS

Figure 7.6 reports the results on the scheduling of an EOS benchmark. The first, second and third columns indicates the name of the instance, the number of variables and the number of constraints, respectively. The fourth column reports the induced width of each instance. The last columns report the cpu time in seconds required by MO–BE and SMO–RDS (i.e., the best approach on these instances so far), respectively. A "-" indicates that the algorithm does not terminate in 1800 seconds. An "out" indicates that

Instance	# vars	# constrs	w^*	Time (sec.)	
				MO–BE	SMO-RDS
1504(0,183)*	184	1329	18	out	1114
1504(184,206)*	23	112	7	0.3	0
1504(356,461)*	106	840	18	out	-
1504(462,508)*	47	301	10	9	0
1506(151,228)	78	1107	24	out	7.8
1407(379,409)*	31	220	11	19	0
1407(413,429)*	17	87	8	0	0
1407(447,469)*	23	129	9	0	0
1407(701,761)	61	445	13	201	3

Figure 7.6: Experimental results on subproblems of the Spot5 instances with capacity constraint and induced width below 24. Time limit 1800 seconds.

MO-BE runs out of memory.

Observe that MO-BE solves instantly instances with small induced width (i.e., below 10). For instances with larger induced width, MO-BE is able to solve them in a reasonable time. However, it is unable to solve three instances (i.e., 1504(0,183)*, 1504(356,461)* and 1506(151,228)) because of the memory limitations. Note that the induced width of these instances is relatively large.

7.5 Related Work

There have been several attempts to describe an abstract inference algorithm able to cope with different reasoning tasks. These descriptions are based on particular algebraic structures that axiomatically describe the combination and marginalization operators. The main examples are the *fusion algorithm* [135] (also studied in [82]), which relies on valuation algebras; the *dynamic programming* approach proposed in [17], based on c-semirings; and the *generalized distributive law* [2] that uses a commutative semiring to describe the properties of both operators. Although non of them are explicitly described to compute the efficient frontier of a multi-objective problem, the instantiation of combination as $+^{mo}$ and marginalization as $\min^{mo}$ leads to

multi-objective inference algorithms very related to our MO-BE.

It is worth noting the clear relation between MO-BE and the adaptation of the well-known *dynamic programming* procedure [16] to the multi-objective case. Mainly, those multi-objective dynamic programming algorithms are restricted to problems with a particular structure such as the shortest path problem, the knapsack problem, the traveling salesperson problem and the transportation problem (we refer the reader to [41] for a complete bibliographical survey on these problems). Our MO-BE is a general algorithm able to compute the efficient frontier of any multi-objective optimization problem described as a graphical model.

7.6 Conclusions

MO-BE is the first general complete variable elimination algorithm for multi-objective optimization tasks. Therefore, it is an alternative approach to solve multi-objective problems.

The theoretical complexity of the algorithm clearly indicates that, as its mono-objective counterpart, multi-objective bucket elimination is suitable for problems with small induced width. We empirically demonstrate that MO-BE can be used to efficiently solve true multi-objective problems with bounded induced width.

We point out that the generalization of MO-BE to solve other multi-objective problems formalized as instances of the semiring CSP framework is straightforward. The structure of the generalized algorithm is the same as for MO-BE. The only difference being the use of the combination and marginalization operators of the given problem.

Finally, it is important to recall that, as described in Section 7.5, AND/OR search with full catching has been proved to be very close related to Bucket Elimination [101, 98, 100, 30]. Its main difference is the order in which new functions are constructed. Concerning the extension of AND/OR search to multi-objective optimization pointed out in Chapter 5, we want to investi-

gate the effect of full catching in the real space needed to store the whole set of current functions. Moreover, we want to extend partial catching schemes [101] to multi-objective optimization in order to have a family of parameterized algorithms that can accommodate trade-offs between time and space.

Chapter 8

Lower Bounding

The importance of algorithms that compute lower bounds is two-fold. On the one hand, they can be used as stand-alone algorithms to approximate the optimum of optimization problems that are too difficult to be solved. On the other, they can be used as bounding evaluation functions inside branch-and-bound algorithms. Many lower bounding algorithms have been proposed in the mono-objective context (see Section 3.1.3). However, very little is known about lower bounding techniques for multi-objective optimization problems.

The purpose of this chapter is to address this lack by extending mini-bucket elimination to multi-objective optimization problems. Mini-bucket elimination is specially convenient for our purposes mainly for two reasons. First, it has a control parameter z that trades time and space for accuracy. Therefore, it is convenient either as a stand-alone approximation algorithm (with large values of z) and as a bounding heuristic inside branch-and-bound (with small values of z). Second, it has been proved to be efficient in many mono-objective reasoning tasks.

The structure of the chapter is as follows. Section 8.1 describes mini-bucket elimination. Then, Section 8.2 presents multi-objective mini-bucket elimination, the extension of mini-bucket elimination to multi-objective optimization. Section 8.3 reports some experimental results using the new algorithm as an approximation of the efficient frontier and as a bounding heuristic

inside multi-objective branch-and-bound. Section 8.4 discusses some related work. Finally, Section 8.5 points out some conclusions.

8.1 Mini-bucket elimination

Mini-Bucket Elimination (MBE) [38] is an approximation algorithm designed to avoid the time and space problem of full bucket elimination (BE). It has a control parameter z that allows us to trade time and space for accuracy. Given a WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ and the value of the control parameter z, MBE computes a lower bound of the optimum of P. In general, higher values of z results in more accuracy lower bounds. In the limit (e.g., when z is the number of variables of the problem) MBE behaves as BE and computes the optimum of P.

As we have seen in Section 7.1, BE solves the original WCSP problem P by eliminating one variable at a time. The result of eliminating variable x_i is a new problem where x_i does not appear in the scope of its set of functions. Given a static variable ordering, the set of functions computed in each problem transformation can be viewed as the following recursion,

$$\mathcal{F}^i = \{\min_{x_i}\{ \sum_{\substack{f \in \mathcal{F}^{i+1} \\ f \in \mathcal{B}_i}} f\}\} \ \cup \ \{f \in \mathcal{F}^{i+1} \mid f \notin \mathcal{B}_i\}$$

from $i = n$ down to 1, where by definition $\mathcal{F}^{n+1} = \mathcal{F}$ and $\mathcal{B}_i = \{f \in \mathcal{F}^{i+1} \mid x_i \in var(f)\}$. The optimum of P is $F^1()$. Note that, in each step i, BE explicitly computes the function $g_i = \min_{x_i}\{\sum_{f \in \mathcal{B}_i} f\}$. The new function g_i represents the same information as the sum of the set of functions in the bucket $\mathcal{B}_i$ but without mentioning x_i. It is easy to see that the computation of g_i is time $O(exp(|var(g_i)|+1))$ and space $O(exp(|var(g_i)|))$, where its arity $|var(g_i)|$ is the joint arity of functions in $\mathcal{B}_i$. Formally,

$$|var(g_i)| = |\bigcup_{f \in \mathcal{B}_i} var(f)|$$

Abusing notation, we denote by $var(\mathcal{B}_i)$ the union of the scopes of the set of functions $\mathcal{B}_i$ and by $|var(\mathcal{B}_i)|$ its joint arity.

MBE also transforms the original WCSP problem P by eliminating one variable at a time. However, the idea is to restrict the effort spent to eliminate the variables of the problem according to the control parameter z. If eliminating variable x_i from $\mathcal{B}_i$ (i.e., computing function g_i) is very expensive in time and/or space such bucket is partitioned into smaller subsets $\{\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_r}\}$, called *mini-buckets*. The joint arity of the functions in each mini-bucket $\mathcal{B}_{i_m}$ is bounded by $z+1$. Then, variable x_i is eliminated from each mini-bucket independently.

The execution of MBE can be described as a recursion,

$$\mathcal{F}^i = \bigcup_{m=1}^{r} \{\min_{x_i}\{ \sum_{f \in \mathcal{B}_{i_m}} f\}\} \ \cup \ \{f \in \mathcal{F}^{i+1} \mid f \notin \mathcal{B}_i\}$$

from $i = n$ down to 1 where, by definition, $\mathcal{F}^{n+1} = \mathcal{F}$ and $\forall\, 1 \leq m \leq r,\ |var(\mathcal{B}_{i_m})| \leq z+1$. Now, in each step of the recursion, MBE explicitly computes a set of functions $g_{i_m} = \min_{x_i}\{\sum_{f \in \mathcal{B}_{i_m}} f\}$ such that $|var(g_{i_m})| \leq z+1$. If there is some function $f \in \mathcal{F}^i$ such that $var(f) > z+1$, then f is discarded.

The partition of each bucket $\mathcal{B}_i$ into a set of mini-buckets $\{\mathcal{B}_{i_1}, \ldots \mathcal{B}_{i_r}\}$ has two main consequences. The first one is that, since the arity of each g_{i_m} is bounded by $z+1$, the space and time complexity for computing each function g_{i_m} is also bounded by z. More precisely, they are $O(exp(z))$ and $O(exp(z+1))$, respectively. Therefore, the space and time complexity of the recursion is also exponential in the control parameter z.

The second consequence is that MBE computes a lower bound of the optimum of the original problem P. Observe that the objective function represented by $\mathcal{F}^i$ in BE is

$$F^i_{BE} = g_i + \sum_{\substack{f \in \mathcal{F}^{i+1} \\ f \notin \mathcal{B}_i}} f$$

while in MBE is

$$F^i_{MBE} = \sum_{m=1}^{r} g_{i_m} + \sum_{\substack{f \in \mathcal{F}^{i+1} \\ f \notin \mathcal{B}_i}} f$$

It can be shown that

$$\forall\, t,\ \overbrace{\min_{x_i}\{\sum_{f \in \mathcal{B}_i} f(t)\}}^{g_i(t)} \geq \sum_{m=1}^{r} \overbrace{\min_{x_i}\{\sum_{f \in \mathcal{B}_{i_m}} f(t)\}}^{g_{i_m}(t)}$$

Then, $\forall\, t,\ F^i_{BE}(t) \geq F^i_{MBE}(t)$. Namely, since the optimal cost of extending t to the eliminated variables $x_i, \ldots, x_n$ is $F^i_{BE}(t)$ and $F^i_{BE}(t) \geq F^i_{MBE}(t)$, MBE computes a lower bound of this extension for each tuple t. As a consequence, the bound computed for each tuple in each step of the recursion results in a lower bound of the optimum of P because $F^1_{BE}() \geq F^1_{MBE}()$.

Example 8.1.1 *Let f and g be two functions defined over the same variable x_i with two domain values a and b. Let $f(a) = 4$, $f(b) = 3$, $g(a) = 1$, and $g(b) = 2$. Consider the elimination of x_i. BE would compute the function $g_i() = \min\{f(a) + g(a), f(b) + g(b)\}$. If MBE splits these two functions in two different mini-buckets, it will compute two zero-arity functions $g_{i_1} = \min\{f(a), f(b)\}$ and $g_{i_2} = \min\{g(a), g(b)\}$. Then, it is easy to see that $g_i() \geq g_{i_1}() + g_{i_2}()$ because $\min\{f(a) + g(a), f(b) + g(b)\} \geq \min\{f(a), f(b)\} + \min\{g(a), g(b)\}$.*

Figure 8.1 shows an algorithmic description of MBE. Its structure is very similar to BE (Figure 7.1). The only difference being that lines 3 and 4 in the BE algorithm are replaced by lines $3-5$ in MBE. The input of MBE is a WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ and the value of the control parameter z. Variables are eliminated one by one, from last to first according to an order. For each variable x_i (line 1), MBE computes its associated bucket $\mathcal{B}_i$ (line 2). Then, it creates a partition $\{\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_r}\}$ of the functions in $\mathcal{B}_i$ (line 3), where $|var(\mathcal{B}_{i_m})| \leq z + 1$. Each $\mathcal{B}_{i_m}$ is processed separately, thus computing a set of functions $\{g_{i_m}\}_{m=1}^{r}$ (line 4). Then, the set of functions

function MBE$((\mathcal{X}, \mathcal{D}, \mathcal{F}), z)$
1. **for each** $i = n..1$ **do**
2. $\quad \mathcal{B}_i := \{f \in F \mid x_i \in var(f)\}$
3. $\quad \{\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_r}\} :=$ Partition$(z, \mathcal{B}_i)$;
4. $\quad$ **for each** $m = 1 \ldots r$ **do** $g_{i_m} := \min_{x_i}\{\sum_{f \in \mathcal{B}_{i_m}} f\}$;
5. $\quad F := (F \cup \{g_{i_1}, \ldots, g_{i_r}\}) - \mathcal{B}_i$;
6. **endfor**
7. **return** $\sum_{f \in \mathcal{F}} f()$;

endfunction

Figure 8.1: Mini-Bucket Elimination algorithm. Given a WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ and the value of the control parameter z, the algorithm returns a lower bound of the optimal cost of P.

$\mathcal{F}$ is updated removing the functions in $\mathcal{B}_i$ and adding the set of functions g_{i_m} (line 5). As we have seen, the bound computed in each bucket $\mathcal{B}_i$, yields MBE to compute a lower bound of the optimum of P. This lower bound is the result of summing the empty-scope functions stored in $\mathcal{F}$ produced when the last variable is eliminated (line 7).

Theorem 8.1.1 *[38] The complexity of* MBE(z) *is time* $O(e \times d^{z+1})$ *and space* $O(e \times d^z)$*, where* e *is the number of functions and* d *is the largest domain size.*

Parameter z allows us to trade time and space for accuracy. In general, greater values of z increment the number of functions included in each mini-bucket. Therefore, the bound will be presumably closer to the cost of the optimal solution. However, the space demands also increase.

Example 8.1.2 *Consider the WCSP instance of example 7.1.1, with seven variables and the following set of cost functions,*

$$\mathcal{F} = \{f_1(x_6, x_5, x_4), f_2(x_6, x_5, x_3), f_3(x_5, x_3, x_2), f_4(x_6, x_4, x_2),$$

$$f_5(x_7, x_2, x_1), f_6(x_7, x_6, x_1)\}$$

One possible execution of MBE(3) *along the lexicographical variable ordering leads to the following trace,*

Buckets	**Mini-buckets**	
$\mathcal{B}_7$	$f_6(x_7, x_6, x_1)$ $f_5(x_7, x_2, x_1)$	
$\mathcal{B}_6$	$f_2(x_6, x_5, x_3)$ $f_1(x_6, x_5, x_4)$	$g_{7_1}(x_6, x_2, x_1) = \min_{x_7}\{f_6 + f_5\}$ $f_4(x_6, x_4, x_2)$
$\mathcal{B}_5$	$g_{6_1}(x_5, x_4, x_3) = \min_{x_6}\{f_1 + f_2\}$ $f_3(x_5, x_3, x_2)$	
$\mathcal{B}_4$	$g_{6_2}(x_4, x_2, x_1) = \min_{x_6}\{f_4 + g_{7_1}\}$ $g_{5_1}(x_4, x_3, x_2) = \min_{x_5}\{f_3 + g_{6_1}\}$	
$\mathcal{B}_3$	$g_{4_1}(x_3, x_2, x_1) = \min_{x_4}\{g_{6_2} + g_{5_1}\}$	
$\mathcal{B}_2$	$g_{3_1}(x_2, x_1) = \min_{x_3}\{g_{4_1}\}$	
$\mathcal{B}_1$	$g_{2_1}(x_1) = \min_{x_2}\{g_{3_1}\}$	
Result	$g_{1_1}() = \min_{x_1}\{g_{2_1}\}$	

The first column indicates the bucket $\mathcal{B}_i$ being treated. Each subsequent column shows the functions included in each mini-bucket $\mathcal{B}_{i_m}$, in case the original bucket needs to be splitted. When the second column is empty, it means that all functions in that bucket can be processed together.

The only bucket splitted into different mini-buckets is $\mathcal{B}_6$. The reason is that the joint arity of $\mathcal{B}_6$ is 6,

$$|var(\mathcal{B}_6)| = |\{x_6, x_5, x_4, x_3, x_2, x_1\}| = 6$$

and the control parameter $z = 3$ restricts the buckets to have a joint arity less than or equal to $3 + 1$. Then, the partition process splits the functions

in $\mathcal{B}_6$ into two mini-buckets, as shown in the previous table. The result are two functions: g_{6_1} and g_{6_2}. The consequence of not processing the functions in $\mathcal{B}_6$ together is that MBE(3) ends up with a lower bound of the optimum of the original problem.

MBE is a powerful mechanism for lower bound computation. It can be used as an stand-alone algorithm to approximate the optimal cost of a difficult problem that cannot be solved exactly. In that case, MBE is executed with the highest possible value of z taking into account the available resources [38]. Moreover, MBE can be used as a bounding evaluation function inside depth-first branch and bound search. In that case, experiments show that low values of z usually provide reasonably good lower bounds with a very low cost [74].

8.2 Multi-objective Mini-Bucket Elimination

Multi-objective mini-bucket elimination (MO-MBE) is the extension of MBE to multi-objective optimization. The idea of the algorithm is the same as for the mono-objective case. Given a MO-WCSP problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, the algorithm sequentially transforms P by eliminating one new variable at a time. Large arity buckets are splitted into smaller mini-buckets in order to bound the effort required to process them according to its control parameter z. As a result, MO-MBE computes a lower bound frontier of the efficient frontier $\mathcal{E}(P)$.

The execution of MO-MBE can be described with the following recursion,

$$\mathcal{F}^i = \bigcup_{m=1}^{r} \{ \min_{x_i}{}^{mo} \{ \sum_{f \in \mathcal{B}_{i_m}}{}^{mo} f \} \} \cup \{ f \in \mathcal{F}^{i+1} \mid f \notin \mathcal{B}_i \} \tag{8.1}$$

from $i = n$ down to 1 where, by definition, $\mathcal{F}^{n+1} = \mathcal{F}$ and $\forall\, 1 \leq m \leq r,\ |var(\mathcal{B}_{i_m})| \leq z + 1$.

This recursion derives two important consequences. The first one is that the space and time complexity of MO-MBE is bounded by z. Let g_{i_m} be the

frontier function computed during the elimination of variable x_i from mini-bucket $\mathcal{B}_{i_m}$. Since the arity of g_{i_m} is bounded by $z+1$, the space and time complexity for computing it is bounded by $exp(z)$ and $exp(z+1)$, respectively (see Theorem 8.2.2 for a detailed description).

The second consequence is that the partition process leads MO-MBE to compute a lower bound frontier of the efficient frontier $\mathcal{E}(P)$. The reason is that, given a tuple t, its valuation in the sum of the set of functions g_{i_m} computed by MBE is a lower bound frontier of its valuation in the function g_i computed by BE. Formally,

$$\forall\, t,\ \overbrace{\min_{a\in D_i}^{mo}\{\sum_{f\in\mathcal{B}_i} f(t\cdot x_i=a)\}}^{g_i(t)} \geq_{mo} \sum_{m=1}^{r} \overbrace{\min_{a\in D_i}^{mo}\{\sum_{f\in\mathcal{B}_{i_m}} f(t\cdot x_i=a)\}}^{g_{i_m}(t)}$$

The intuition behind this inequality is as follows. Given a tuple t, $g_i(t)$ is a frontier representing the non-dominated cost vectors resulting from extending t to the eliminated variable x_i. Then, each cost vector $\vec{v}\in g_i(t)$ must come from a tuple $t\cdot(x_i=a)$. Formally,

$$\vec{v}=\sum_{f\in\mathcal{B}_i} f(t\cdot x_i=a)$$

Therefore, $\vec{v}\in\sum_{m=1}^{r}\sum_{f\in\mathcal{B}_{i_m}} f(t\cdot x_i=a)$. Since in each mini-bucket the value assigned to x_i should be different, either $\vec{v}$ or $\vec{v}'$ that dominates $\vec{v}$ is an element of $\sum_{m=1}^{r} g_{i_m}(t)$. As a result, each cost vector in $g_i(t)$ is dominated by at least one cost vector in $\sum_{m=1}^{r} g_{i_m}(t)$, which is exactly the non-domination partial order among frontiers $\geq_{mo}$. We say that a frontier function f is a lower bound of function h when, $\forall\, t,\ f(t)\leq_{mo} h(t)$. Then, $\sum_{m=1}^{r} g_{i_m}$ is a lower bound of g_i. It is easy to see that $\leq_{mo}$ satisfies transitivity. Since MO-MBE processes buckets where all functions are either original or recursively processed by MO-MBE (which are lower bounds themselves), all functions computed by MO-MBE in a bucket are lower bounds of the function that MO-BE would compute at that bucket.

Figure 8.2 shows an algorithmic description of MO-MBE. It receives a MO-WCSP problem $P=(\mathcal{X},\mathcal{D},\mathcal{F})$ and the value of the control parameter

function MO-MBE$((\mathcal{X}, \mathcal{D}, \mathcal{F}), z)$
1. **for each** $i = n \dots 1$ **do**
2. $\quad \mathcal{B}_i := \{f \in \mathcal{F} \mid x_i \in var(f)\}$;
3. $\quad \{\mathcal{B}_{i_1}, \dots, \mathcal{B}_{i_r}\}$:=Partition$(z, \mathcal{B}_i)$;
4. $\quad$ **for each** $m = 1 \dots r$ **do** $g_{i_m} := \min_{x_i}^{mo}\{\sum_{f \in \mathcal{B}_{i_m}} f\}$;
5. $\quad \mathcal{F} := (\mathcal{F} \cup \{g_{i_1}, \dots, g_{i_r}\}) - \mathcal{B}_i$;
6. **endfor**
7. **return** $\sum_{f \in \mathcal{F}}^{mo} f()$;

endfunction

Figure 8.2: Description of MO-MBE. The input is a MO-WCSP instance $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ and the value of the control parameter z. The output is a lower bound frontier of the efficient frontier $\mathcal{E}(P)$.

z. Since the algorithm deals with frontier functions, the combination and marginalization operators are the ones defined over this type of functions. In each iteration, the algorithm eliminates one new variable according to a given order (that we assume lexicographic without loss of generality). When eliminating variable x_i (line 1), MO-MBE computes its bucket $\mathcal{B}_i$ (line 2) and a partition $\{\mathcal{B}_{i_1}, \dots, \mathcal{B}_{i_r}\}$ such that $|var(\mathcal{B}_{i_m})| \leq z + 1$. Then, the algorithm computes a set of functions g_{i_m} resulting from the sum of functions in $\mathcal{B}_{i_m}$ and the subsequent elimination of x_i. The set of functions $\mathcal{F}$ is updated removing the set of functions in $\mathcal{B}_i$ and adding the new functions g_{i_m}. The elimination of the last variable results in a set of zero-arity frontier functions stored in $\mathcal{F}$. The combination of such functions results in a set of non-dominated cost vectors, which is a lower bound frontier of P.

Theorem 8.2.1 *MO-MBE computes a lower bound frontier of the original MO-WCSP problem.*

Example 8.2.1 *Consider the MO-WCSP problem P of Example 7.3.1. The frontier functions are:*

$$h_1(x_1, x_3) = \begin{cases} \{(0,0,0)\} & (x_1 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \\ \top & otherwise \end{cases}$$

$$h_2(x_3, x_4) = \begin{cases} \{(0,0,0)\} & x_3 \vee x_4 \\ \top & otherwise \end{cases}$$

$$h_3(x_2, x_4) = \begin{cases} \{(0,0,0)\} & \overline{x_2 \wedge x_4} \\ \top & otherwise \end{cases}$$

$$p_i(x_i) = \begin{cases} \{(0,0,0)\} & x_i = 1 \\ \{(i,0,0)\} & x_i = 0 \end{cases}$$

$$p_{23}(x_2, x_3) = \begin{cases} \{(0,0,0)\} & x_2 \wedge x_3 \\ \{(3,0,0)\} & otherwise \end{cases}$$

$$w_i(x_i) = \begin{cases} \{(0,5-i,0)\} & x_i = 1 \\ \{(0,0,0)\} & x_i = 0 \end{cases}$$

$$v_i(x_i) = \begin{cases} \{(0,0,i)\} & x_i = 1 \\ \{(0,0,0)\} & x_i = 0 \end{cases}$$

where the top valuation indicating forbidden assignments is $\top = \{(\infty, \infty, \infty)\}$.

The trace of MBE(1) under lexicographical ordering is:

- *Input: the MO-WCSP problem P and the value of the control parameter $z = 1$.*
- *Elimination of x_4: $\mathcal{B}_4 = \{h_2, h_3, p_4, w_4, v_4\}$. Their arity is 3. Therefore, $\mathcal{B}_4$ cannot be jointly processed, but must be splitted into smaller subsets.*

Let $\mathcal{B}_{4_1} = \{h_2, p_4, w_4\}$ and $\mathcal{B}_{4_1} = \{h_3, v_4\}$ be the two resulting mini-buckets. Their sum is $b_{4_1}(x_3, x_4)$ and $b_{4_2}(x_2, x_4)$, respectively,

$$\begin{array}{ll} b_{4_1}(00) = \top & b_{4_2}(00) = \{(0,0,0)\} \\ b_{4_1}(01) = \{(0,1,0)\} & b_{4_2}(01) = \{(0,0,4)\} \\ b_{4_1}(10) = \{(4,0,0)\} & b_{4_2}(10) = \{(0,0,0)\} \\ b_{4_1}(11) = \{(0,1,0)\} & b_{4_2}(11) = \top \end{array}$$

Note that, the $\top$ valuation is dominated by any other. As a result, only forbidden assignments will take that valuation.

Projecting x_4 out of b_{4_1} and b_{4_2} produces $g_{4_1}(x_3)$ and $g_{4_2}(x_2)$ respectively,

$$\begin{array}{ll} g_{4_1}(0) = \{(0,1,0)\} & g_{4_1}(1) = \{(4,0,0),(0,1,0)\} \\ g_{4_2}(0) = \{(0,0,4)\} & g_{4_2}(1) = \{(0,0,0)\} \end{array}$$

- *Elimination of x_3: $\mathcal{B}_3 = \{g_4, h_1, p_3, p_{23}, w_3, v_3\}$. The arity of $\mathcal{B}_3$ is 3. Therefore, it is split into two mini-buckets $\mathcal{B}_{3_1} = \{g_{4_1}, h_1, p_3\}$ and $\mathcal{B}_{3_1} = \{p_{2_3}, w_3, v_3\}$. Their sum is $b_{3_1}(x_1, x_3)$ and $b_{3_2}(x_2, x_3)$ respectively,*

$$\begin{array}{ll} b_{3_1}(00) = \top & b_{3_2}(00) = \{(3,0,0)\} \\ b_{3_1}(01) = \{(4,0,0),(0,1,0)\} & b_{3_2}(01) = \{(3,2,3)\} \\ b_{3_1}(10) = \{(3,4,0)\} & b_{3_2}(10) = \{(3,0,0)\} \\ b_{3_1}(11) = \top & b_{3_2}(11) = \{(0,2,3)\} \end{array}$$

Projecting x_3 out of b_{3_1} and b_{3_2} produces $g_{3_1}(x_1)$ and $g_{3_2}(x_2)$ respectively,

$$\begin{array}{ll} g_{3_1}(0) = \{(4,0,0),(0,1,0)\} & g_{3_2}(0) = \{(3,0,0)\} \\ g_{3_1}(1) = \{(3,4,0)\} & g_{3_2}(1) = \{(3,0,0),(0,2,3)\} \end{array}$$

- *Elimination of x_2: $\mathcal{B}_2 = \{g_{4_2}, g_{3_2}, p_2, w_2, v_2\}$. Their arity is 1. Therefore, they can be jointly processed. Their sum is $b_2(x_2)$,*

$$b_2(0) = \{(5,0,4)\} \quad b_2(1) = \{(3,3,2),(0,5,5)\}$$

Projecting x_2 out of b_2 produces $g_2()$,

$$g_2() = \{(5, 0, 4), (3, 3, 2), (0, 5, 5)\}$$

Note that $g_2()$ is a zero-arity function. Its frontier will be summed to the one obtained when eliminating x_1 from the problem.

- *Elimination of x_1: $\mathcal{B}_1 = \{g_{3_1}, p_1, w_1, v_1\}$. Their arity is 1. Therefore, they can be jointly processed. Their sum is $b_1(x_1)$,*

$$b_1(0) = \{(5, 0, 0), (1, 1, 0)\} \qquad b_1(1) = \{(3, 8, 1)\}$$

Projecting x_1 out of b_1 produces $g_1 = \{(5, 0, 0), (1, 1, 0)\}$

Note that $(3, 8, 1)$ is not a valid cost vector as it is dominated by $(1, 1, 0)$.

- *Output: the final lower bound frontier is computed summing all the zero-arity functions obtained during the process. Hence, the lower bound frontier returned is,*

$$\texttt{lbf} = \{(1, 6, 5), (4, 4, 2), (6, 1, 4), (8, 3, 2), (10, 0, 4)\}$$

Recall that the efficient frontier of the problem is

$$\mathcal{E}(P) = \{(10, 2, 3), (6, 3, 7), (5, 5, 5)\}$$

Note that `lbf` *is a lower bound frontier of $\mathcal{E}(P)$ because* `lbf` $\leq_{mo} \mathcal{E}(P)$.

Theorem 8.2.2 *MO-MBE with control parameter z is space $O(e \times \prod_{j=1}^{p-1} K_j \times d^z)$ and time $O(e \times \prod_{j=1}^{p-1} K_j^2 \times d^{z+1})$, where e is the number of frontier functions, K_j is the maximum possible cost of objective j (as described in Observation 7.3.1), p is the number of objectives, and d is the largest domain size.*

Proof The structure of the proof is the same as for the complexity of complete bucket elimination (see Theorem 7.3.1). In this case, as the arity of new functions is bounded by the control parameter z, the time and space complexity directly holds. ∎

8.3 Experimental Results

We analyze the suitability of MO-MBE either as a bounding evaluation function inside MO-BB (described in Section 5.2) and as a stand-alone approximation algorithm. To that end, we structure this section in two parts, one for each potential use of MO-MBE.

We test the algorithms in four different domains: *Max-SAT-ONE*, *biobjective minimum vertex cover*, *risk-conscious combinatorial auctions* and *scheduling of an EOS* (for a detailed description see Appendices B.4, B.4, B.1 and B.2, respectively).

It is worth noting that we consider the induced width of all instances under the variable order given by the min-fill heuristic. Recall that the min-fill heuristic ensures variable orderings with small induced width. MO-MBE follows this elimination order in all the experiments .

The time limit for all the experiments is 1800 seconds.

8.3.1 MO-MBE as a bounding heuristic function

The purpose of these experiments is to evaluate the performance of MO-MBE as a bounding heuristic function inside MO-BB (i.e. MO-BB$_{mombe}$). This evaluation has two aspects:

- The effect of the control parameter z in the performance of MO-BB$_{mombe}$. As noted in Section 8.2, as the value of the control parameter z increases, the accuracy of the lower bound frontier computed by MO-MBE will probably increase. At the same time, the cpu time required to compute the lower bound also increases. Therefore, there is a trade-off between the time to compute the lower bound frontier and its accuracy. Experiments in the mono-objective case show that low values of the control parameter z usually provide reasonable good lower bounds with a very low cost [74]. To give evidence of this hypothesis also in the multi-objective case, we experiment with three values of z (i.e., 2, 4 and 8) in the different domains.

- The overall performance of MO-BB$_{mombe}$ with respect to other heuristic functions used within MO-BB. To that end, we compare MO-BB$_{mombe}$ versus the best approach between MO-BB$_{\texttt{icf}}$, MO-BB$_{fdac}$ and (S)MO-RDS.

As we have seen in the previous Chapter, MO-BE is very efficient in instances with induced width below 24. Therefore, we experiment on instances that either could not be solved for space limitations or could not be efficiently solved by MO-BE. Moreover, we disregard the time spent by ϵ-constraint since, as seen in the previous Chapters, it is always outperformed.

Max-SAT-ONE

Figure 8.3 shows the results for Max-SAT-ONE instances not very convenient for MO-BE. The first column contains the instance name. The second column shows the size of the efficient frontier. The third column contains the induced width obtained by the min-fill heuristic of each instance. The three following columns contain the cpu time in seconds required by MO-BB$_{mombe}$ setting the control parameter z to 2, 4 and 8, respectively. The last two columns indicate the cpu time in seconds required by MO-BB$_{\texttt{icf}}$ and MO-BB$_{fdac}$. We disregard the time required by (S)MO-RDS because we have showed that it was outperformed by the previous two approaches. A "-" indicates that the algorithm does not terminate in 1800 seconds.

Regarding the relative performance of MO-BB$_{mombe}$ with respect to the value of the control parameter z, the best results are obtained with $z = 4$. These results are consistent with the hypothesis that relative low values of z compute good lower bounds in reasonable time.

In general, MO-BB$_{mombe}$ is not able to solve instances with very high induced width (see *aim-50-6_0* instances). In this case, MO-MBE partitions each bucket in a large number of mini-buckets. The main consequence is that the lower bound frontier computed by MO-MBE is not very accurate and the pruning capability of MO-BB is very low.

Instance	$\|\mathcal{E}\|$	w*	MO-BB$_{mombe}$			MO-BB$_{\texttt{icf}}$	MO-BB$_{fdac}$
			$z=2$	$z=4$	$z=8$		
aim-50-2_0-no-1	12	26	395.14	321.53	560.19	67.35	319.44
aim-50-2_0-no-2	10	26	155.86	58.26	172.37	45.61	87.88
aim-50-2_0-no-3	10	26	93.1	71.94	285.23	17.06	32.08
aim-50-2_0-no-4	10	23	133.66	118.68	412.5	25.2	70.05
aim-50-2_0-yes1-1	14	25	441.3	195.65	309.45	97.99	725.26
aim-50-2_0-yes1-2	12	23	244.42	143.02	261.88	69.68	345.93
aim-50-2_0-yes1-3	14	23	339.48	165.37	205.6	72.11	443.72
aim-50-3_4-yes1-1	15	32	426.28	323.76	1061.42	71.06	211.56
aim-50-3_4-yes1-2	17	31	1147.26	719.41	-	199.91	520.62
aim-50-3_4-yes1-3	19	31	-	1214.97	-	309.71	1535.47
aim-50-3_4-yes1-4	19	31	1154.57	719.3	1769.85	184.12	900.66
aim-50-6_0-yes1-1	27	37	-	-	-	1475.08	-
aim-50-6_0-yes1-2	26	37	-	-	-	1525.85	-
aim-50-6_0-yes1-3	23	37	-	-	-	791.3	-
aim-50-6_0-yes1-4	23	36	-	-	-	690.42	-

Figure 8.3: Experimental results on Max-SAT-ONE problems. Time limit 1800 seconds.

MO-BB$_{\texttt{icf}}$ seems to be the most efficient approach in this benchmark. Note that both $\texttt{lbf}_{\texttt{icf}}$ and the lower bound computed by MO-MBE are multi-objective. However, the first one is simpler and computed more efficiently.

Biobjective Minimum Vertex Cover

Figure 8.4 reports the results obtained for biobjective weighted minimum vertex cover instances that could not be solved with MO-BE. The first and second columns show the number of variables and edges, respectively. The third and fourth columns report the mean size of the efficient frontier and the mean induced width of each parameter configuration. The three next columns indicate the mean cpu time in seconds required by MO-BB$_{mombe}$ setting the control parameter z to 2, 4 and 8, respectively. The eighth and ninth columns show the mean cpu time in seconds required by MO-BB$_{\texttt{icf}}$ and MO-BB$_{fdac}$, respectively. Moreover, we parenthesize the percentage of solved instances within the time limit when it is different from 100%. We do not report the results obtained by (S)MO-RDS because, as shown in Chapter 6, it is not very suitable in this benchmark.

N (#vars)	E (#edges)	$\|\mathcal{E}\|$	w*	MO-BB$_{mombe}$			MO-BB$_{\texttt{icf}}$	MO-BB$_{fdac}$
				$z=2$	$z=4$	$z=8$		
60	250	5,04	27	1,91	2,57	17,72	1,70	0,3276
70		6,68	26,9	4,44	5,24	35,77	16,07	1,69
80		8,6	27,3	9,53	9,59	57,91	110,58	10,67
90		8,76	29,2	15,46	13,38	71,39	698,59 (96%)	53,95
60	500	4,72	39,9	2,24	3,81	32,32	0,36	0,11
70		5,4	41,3	6,49	10,21	86,65	3,03	0,55
80		6,6	45,6	18,70	24,90	208,93	16,97	2,93
90		7,76	47	50,24	56,80	437,17	117,463	15,78
60	950	3,76	48,3	1,80	3,04	20,93	0,082	0,03
70		4,04	53,7	4,40	7,72	71,90	0,49	0,13
80		5,92	57,7	11,79	21,51	193,77	2,00	0,53
90		5,56	63,2	35,53	57,43	548,08	11,50	2,63

Figure 8.4: Experimental results on biobjective weighted minimum vertex cover problems. Parameter C is set to 5. Mean values on 25 instances for each parameter configuration. Time limit 1800 seconds.

Regarding the relative performance of MO-BB$_{mombe}$ with respect to the value of the control parameter z, the best results are obtained with $z = 2$. The only exception is in instances with 90 variables and 250 edges, where MO-BB$_{mombe}$ setting z to 4 is slightly better. It is worth noting the clear difference between the cpu time required by this algorithm using $z = 2$ and $z = 8$. These results confirm also in this benchmark the hypothesis that relative low values of z computes good lower bounds in reasonable time.

If we compare the relative performance of MO-BB with respect to the different lower bounds used, it seems that MO-BB$_{mombe}$ is the most efficient approach for loose parameter configurations (i.e., with small ratio between variables and constraints). On the other hand, it seems that MO-BB$_{fdac}$ is the most efficient approach for tight parameter configurations (i.e., with high ratio between variables and constraints).

Combinatorial Auctions

Figure 8.5 reports the results obtained for risk-conscious auctions instances with 20 (left) and 50 goods (right). We report mean cpu time (top) and mean

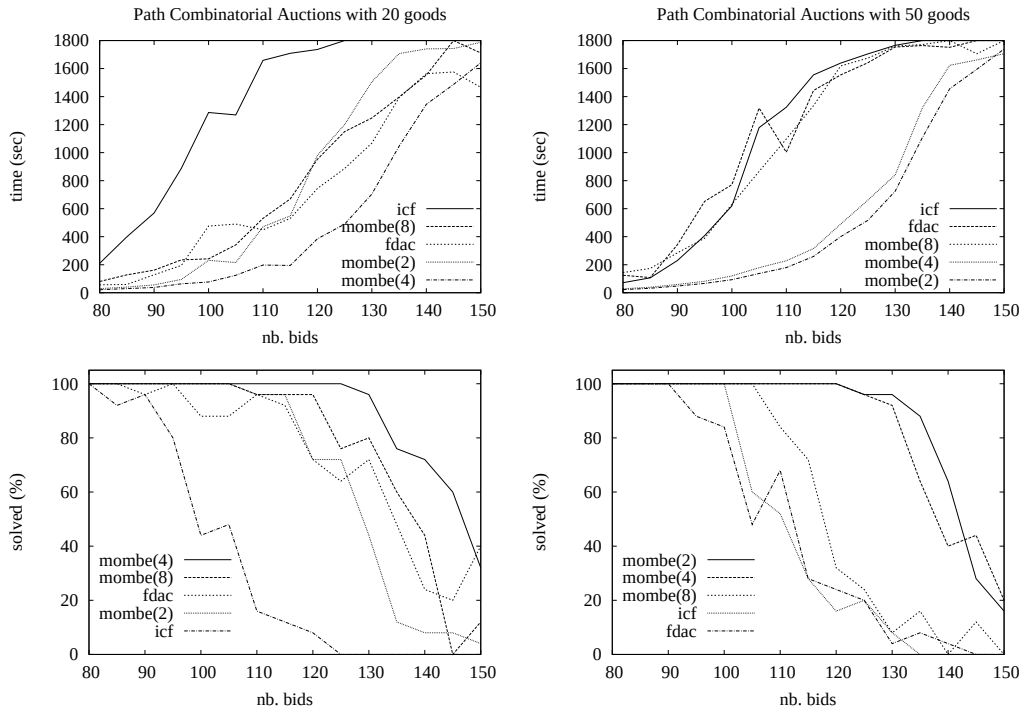

Figure 8.5: Experimental results on risk-conscious combinatorial auctions for 20 and 50 goods, respectively. Path distribution. Time limit 1800 seconds.

solved percentage within the time limit (bottom). We consider the time limit as the cpu time for unsolved instances. MO-BB$_{mombe}$ is only compared with MO-BB$_{\texttt{icf}}$ and MO-BB$_{fdac}$ because, as shown in the previous chapters, the other approaches are clearly worst. For that reason, we only indicate the lower bound used in the key of the previous figure.

The relative performance of MO-BB$_{mombe}$ with respect to the value of the control parameter z depends on the number of goods. For instances with 20 goods, the best results are obtained with $z = 4$. For instances with 50 goods, the best results are obtained with $z = 2$. It is again empirically demonstrated that lower values of z compute relative good lower bounds.

MO-BB$_{mombe}$ is the best approach in both configurations. It is the only

Instance	var	constr	MO-BB$_{mombe}$			(S)MO-RDS	MO-BB$_{\texttt{icf}}$
			$z=2$	$z=4$	$z=8$		
1504(0,183)*	184	1329	-	-	-	1114	-
1506(0,150)	151	1440	-	-	-	-	-
1506(151,228)	78	1107	-	-	-	7.8	-
1506(229,317)	89	1349	-	-	-	62	-
1506(679,761)*	83	1243	-	-	-	1	-
1405(762,854)*	93	2193	-	-	-	1.58	-
1407(0,147)	148	1442	-	-	-	1625	-
1407(148,247)	100	1678	-	-	-	17.12	-
1407(248,378)	131	3063	-	-	-	366	-
1407(413,429)*	17	87	0.16	0.2	1.31	0	0.02
1407(447,469)*	23	129	0.03	0.13	0.13	0	0
1407(494,553)*	60	1333	-	-	-	267	-
1407(580,700)	121	2299	-	-	-	1769	-
1407(762,878)*	117	2708	-	-	-	4.8	-

Figure 8.6: Experimental results on subproblems of the Spot5 instances with capacity constraint. Time limit 1800 seconds.

approach able to solve all instances up to 125 and 120 bids with 20 and 50 goods, respectively. The greatest gap in the cpu time between MO-BB$_{mombe}$ and MO-BB$_{fdac}$ (i.e., the second best approach) occurs in instances with 50 goods. Note that the performance of MO-BB$_{mombe}$ is almost the same in both configurations. However, the performance of MO-BB$_{fdac}$ decreases in instances with 50 goods.

Scheduling of an Earth Observation Satellite

Figure 8.6 reports the results on the scheduling of an EOS benchmark. The first column of the figure indicates the name of the instance. The second and third columns indicate the number of variables and constraints, respectively. The three following columns show the cpu time in seconds required by MO-BB$_{mombe}$ setting the control parameter z to 2, 4 and 8, respectively. The remaining columns indicate the same information for (S)MO-RDS and MO-BB$_{\texttt{icf}}$. A "-" indicates that the algorithm cannot solve the subproblem within the time limit.

The main observation is that MO-BB$_{mombe}$ is only able to solve two in-

stances within the time limit. Note that it solves the same instances as MO-BB$_{\texttt{icf}}$. These results confirm the hypothesis stated in Chapter 5 that MO-BB is not suitable for this kind of instances. As we have seen in the previous chapter, (S)MO-RDS seems to be the proper approach in this benchmark.

8.3.2 MO-MBE as a pre-process

The purpose of these experiments is to evaluate the trade-off between accuracy and efficiency of MO-MBE as an approximation algorithm. To that end, we compare MO-MBE with a modification of the ϵ-constraint approach to compute a lower bound frontier of a given problem. This algorithm, called *ϵ-constraint-lb*, is depicted in Figure 8.7. As usual, we use IlogSolver 6.1 as solver engine. Moreover, we run the algorithm with l_1 and l_2 being the optimal cost of each objective function when considered independently, and the time spent to compute them are not taken into account.

The efficient frontier of bi-objective problems define a region in the 2-dimensional space. One usual way to compare the quality of one bound with respect to the other is to compute the ratio of their areas.

In this section we disregard the risk conscious combinatorial auctions benchmark. As seen in Section 5.3, these instances are very difficult for ϵ-constraint. In particular, this algorithm is only able to solve some instances with 20 goods and less than 100 bids. We empirically corroborated that the behaviour of ϵ-constraint-lb is quite similar. As a consequence, ϵ-constraint-lb is not an appropriate alternative in this case.

Max-SAT-ONE

Figure 8.8 and Figure 8.9 reports the results obtained for those Max-SAT-ONE instances that have not been solved by any exact multi-objective algorithm in the previous Chapters. The sixth and eighth columns show the size of the lower bound frontier found by MO-MBE (for each configuration of parameter z) and ϵ-constraint-lb, respectively. The tenth column reports

function ϵ-`constraint-lb` $((\mathcal{X}, \mathcal{D}, \mathcal{F}), (l_1, l_2), (u_1, u_2)))$
return frontier
1. $\mathcal{E} := \emptyset$; `lbf` $:= \{(l_1, l_2)\}$;
2. $i := l_2 + 1$;
3. **while** $i \leq u_2 + u_1 - l_1$ **do**
4. $\quad \epsilon_1 := l_1 + \max(0, i - u_2)$;
5. $\quad \epsilon_2 := \min(i, u_2)$;
6. $\quad$ **while** $\epsilon_2 \geq l_2$ **and** $\epsilon_1 \leq u2$ **do**
7. $\quad\quad$ **if** $\mathcal{E} \not\leq \{(\epsilon_1, \epsilon_2)\}$ **then**
8. $\quad\quad\quad$ **if** solve $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ subject to $F_1 \leq \epsilon_1$ **and** $F_2 \leq \epsilon_2$ **then**
9. $\quad\quad\quad\quad \mathcal{E} := \mathcal{E} \cup \{(\epsilon_1, \epsilon_2)\}$;
10. $\quad\quad\quad$ **else** `lbf` $:= \min^{mo}\{$`lbf`$, \{(\epsilon_1, \epsilon_2)\}\}$;
11. $\quad\quad$ **endif**
12. $\quad\quad \epsilon_1 := \epsilon_1 + 1$;
13. $\quad\quad \epsilon_2 := \epsilon_2 - 1$;
14. $\quad$ **endwhile**
15. $\quad i := i + 1$;
16. **endwhile**
17. **return** $\mathcal{E} \cup$ `lbf`;

endfunction

Figure 8.7: ϵ-constraint-lb algorithm. Given a biobjective MO-WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, the algorithm returns a lower bound frontier of P. Note that if `lbf` $= \emptyset$, $\mathcal{E}$ is the efficient frontier of P. Moreover, if `lbf` $\neq \emptyset$, $\mathcal{E}$ contains part of the efficient frontier and `lbf` is a lower bound frontier of the efficient frontier not contained in $\mathcal{E}$.

Instance	nb. vars	nb. clauses	w^*	MO-MBE(z)			ϵ-constraint-lb		
				z	|lbf|	time (sec.)	|lbf|	time (sec.)	Ratio
aim-100-1-6-no-1	100	160	38	15	10	4.3	7	1800	1.5
				20	12	254.31			2.19
aim-100-1-6-no-2	100	160	39	15	9	3.82	7	1800	1.15
				20	13	232.89			2.68
aim-100-1-6-no-3	100	157	40	15	11	4.11	7	1800	1.75
				20	13	180.17			2.72
aim-100-1-6-no-4	100	160	40	15	15	3.88	7	1800	3.28
				20	11	203.34			1.75
aim-100-1-6-yes1-1	100	160	37	15	15	5.6	7	1800	3.31
				20	15	276.34			3.47
aim-100-1-6-yes1-2	100	160	37	15	12	5.3	6	1800	2.57
				20	14	199.24			3.34
aim-100-1-6-yes1-3	100	160	35	15	13	5.41	7	1800	2.59
				20	13	229.05			2.47
aim-100-1-6-yes1-4	100	160	35	15	14	4.3	6	1800	3.62
				20	17	181.87			5.27
aim-100-2-0-no-1	100	200	48	15	9	4.11	7	1800	1.34
				20	9	200.26			4.41
aim-100-2-0-no-2	100	199	53	15	10	3.67	6	1800	1.77
				20	8	193.93			1.19
aim-100-2-0-no-3	100	198	50	15	5	3.72	7	1800	0.41
				20	8	235.93			0.97
aim-100-2-0-no-4	100	200	48	15	7	4.06	7	1800	0.75
				20	6	197.7			0.5
aim-100-2-0-yes1-1	100	198	45	15	16	3.91	7	1800	4.06
				20	17	295.23			5.19
aim-100-2-0-yes1-1	100	198	45	15	16	3.9	7	1800	4.06
				20	17	215.47			5.19
aim-100-2-0-yes1-3	100	200	47	15	14	4.32	6	1800	3.27
				20	16	284.33			4.93
aim-100-2-0-yes1-4	100	199	47	15	12	4.89	7	1800	2.25
				20	16	308.9			4.5
aim-200-1-6-no-1	200	320	89	15	7	6.58	5	1800	1.26
				20	12	322.36			3.84
aim-200-1-6-no-2	200	317	85	15	13	6.69	6	1800	3
				20	10	361.58			1.77
aim-200-1-6-no-3	200	320	81	15	12	5.96	5	1800	3.47
				20	14	238.95			5.11
aim-200-1-6-no-4	200	320	85	15	8	5.67	6	1800	1.19
				20	12	290.13			2.77
aim-200-1-6-yes1-1	200	320	72	15	17	10.89	6	1800	5.71
				20	23	332.5			10.75
aim-200-1-6-yes1-2	200	320	66	15	18	7.46	6	1800	5.54
				20	20	435.72			6.76
aim-200-1-6-yes1-3	200	319	68	15	25	10.07	6	1800	11.69
				20	28	420.72			14.69
aim-200-1-6-yes1-4	200	320	73	15	19	7.5	6	1800	6.81
				20	21	472.75			8.57
aim-200-2-0-yes1-3	200	399	91	15	17	10.07	6	1800	6.09
				20	21	364.33			8.92

Figure 8.8: Experimental results on Max-SAT-ONE problems. Time limit 1800 seconds.

Instance	nb. vars	nb. clauses	w^*	MO-MBE(z)			ϵ-constraint-lb		Ratio
				z	|lbf|	time (sec.)	|lbf|	time (sec.)	
ssa2670-130	1359	3321	27	10	342	54.33	4	1800	7776.67
				15	349	1510.31			8343.78
ssa7552-038	1501	3575	29	5	254	15.65	4	1800	4185.67
				10	357	128.32			8246.56

Figure 8.9: Experimental results on Max-SAT-ONE problems. Time limit 1800 seconds.

the ratio between the area covered by MO-MBE(z) for different values of z with respect to the area covered by ϵ-constraint-lb (i.e., area MO-MBE(z) / area ϵ-constraint-lb).

For *aim* instances, the first thing to be observed is that the lowest value of z (i.e., $z = 15$) outperforms the approximations given by ϵ-constraint-lb for almost all instances. Note that the time spent by MO-MBE(15) is less than 8 seconds for all those instances while ϵ-constraint-lb reaches the time limit of 1800 seconds. Increasing the value of z allows MO-MBE to compute much more accurate lower bound frontiers and therefore, the ratio also increases.

For *ssa* instances, the advantage of MO-MBE over ϵ-constraint-lb is even greater. MO-MBE is up to 7776.67 times better than ϵ-constraint-lb with the lowest value of z. As before, the ratio increases with highest values of z.

Biobjective Weighted Vertex Cover

For the second domain, we test on samples for the following parameter configurations

$$(\{60, 70, 80, 90\}, \{500, 950\}, 5)$$

Note that the induced width of these instances is the highest and, as a consequence, these instances are difficult for MO-MBE.

Figure 8.10 reports the results obtained for MO-MBE for different values of the accuracy parameter z. The sixth column shows the ratio between the area covered by the lower bound frontier found by MO-MBE(z) and ϵ-constraint-lb.

N (nb. vars)	E (nb. edges)	w^*	MO-MBE z	MO-MBE time (sec.)	Ratio
60	500	39.9	15	0.6	1.36
			20	31.95	1.74
			25	891.48	2.23
70	500	41.3	15	0.58	1.66
			20	38.22	2.20
			25	1040.63	2.36
80	500	45.6	15	0.8	2.60
			20	35.59	3.28
			25	965.49	3.54
90	500	47	15	1.17	3.22
			20	42.8	3.90
			25	1227.2	4.26
60	950	48.35	15	0.33	1.59
			20	12.37	2.14
			25	477.5	2.65
70	950	53.75	15	0.35	1.75
			20	15.53	2.47
			25	772.8	3.08
80	950	57.75	15	0.39	1.81
			20	25.9	2.78
			25	801.46	3.43
90	950	63.25	15	0.51	2.07
			20	23.44	2.96
			25	1043.77	3.82

Figure 8.10: Experimental results on biobjective weighted minimum vertex cover problems. Parameter C is set to 5. Mean values on 25 instances for each parameter configuration. Time limit 1800 seconds.

As can be observed, MO-MBE with the lowest value of z (i.e. $z = 15$) outperforms ϵ-constraint-lb for all parameter configurations. Note that ϵ-constraint-lb reaches the time limit for all instances, whereas the time spent by MO-MBE(15) is less than 2 seconds.

Scheduling of an Earth Observation Satellite

Figure 8.11 shows the results obtained in the scheduling of an EOS benchmark. We experiment on Spot5 instances with high induced width, namely, those instances that were not solved by MO-BE (see Section 7.4). The sev-

Instance	#var	#constr	w*	MO-BB$_{mombe}$		Ratio
				z	time (sec.)	
1504(0,183)*	184	1329	18	10	1.08	33.3
				15	18.26	35.20
				20	544.72	35.49
1504(356,461)*	106	840	18	10	0.14	27.35
				15	5.95	30.358
				20	144.81	34.66
1506(0,150)	151	1440	31	10	0.49	33.06
				15	12.84	36.14
				20	280.42	39.55
1506(229,317)	89	1349	34	10	0.51	12.77
				15	25.7	13.79
				20	753.64	14.91
1506(679,761)*	83	1243	28	10	0.42	12.13
				15	6.85	12.46
				20	424.42	14.73
1405(762,854)*	93	2193	34	10	0.43	14.90
				15	6.19	15.61
				20	591.77	16.70
1407(0,147)	148	1442	29	10	0.54	34.04
				15	15.91	35.64
				20	547.22	38.45
1407(148,247)	100	1678	31	10	0.47	19.17
				15	21.7	20.51
				20	760.3	24.22
1407(248,378)	131	3063	52	10	1.04	23.96
				15	26.49	24.90
				20	1471.78	26.54
1407(494,553)*	60	1333	32	10	0.09	0.28
				15	2.34	0.53
				20	67.06	0.64
1407(580,700)	121	445	44	10	0.17	0.57
				15	4.42	0.64
				20	144.41	0.73
1407(762,878)*	117	2708	34	10	0.62	19.96
				15	11.77	20.68
				20	717.88	22.91

Figure 8.11: Experimental results on subproblems of the Spot5 instances with capacity constraint. Time limit 1800 seconds.

enth column shows the ratio between the area covered by the lower bound frontier found by MO-MBE(z) and ϵ-constraint-lb. The time spent by ϵ-constraint-lb is 1800 seconds for all instances.

In general, MO-MBE obtains much more accurate lower bound frontiers than ϵ-constraint-lb. In particular, MO-MBE is from 12 to 34 times better than ϵ-constraint-lb with the lowest value of z (i.e., $z = 10$). The only exceptions are 1407(494,553)* and 1407(580,700) instances. The efficient frontier of these instances has size 1. In this case recall that, as seen in Section 5.3, the vector (l_1, l_2) received by ϵ-constraint-lb is the efficient cost vector. Under these circumstances, ϵ-constraint-lb computes a lower bound frontier that contains the efficient frontier. The main consequence is that this lower bound frontier is more accurate than the one computed by MO-MBE. Finally, observe that the accuracy of MO-MBE, as well as its cpu time, increases as the value of z increases.

As we have seen in B.2, although the original formulation of Spot5 instances is mono-objective, it can be approximated by a multi-objective approximation algorithm. The lower bound of the original mono-objective problem is the efficient solution of the lower bound frontier of the bi-objective version such that the value of the second objective function does not surpass the available memory and the cost of the first objective is minimum. Hence, we assess the applicability of MO-MBE under these circumstances.

One possible way to compute lower bounds in the original formulation of Spot5 consists on removing the capacity constraint from the instances (the optimum of this relaxation will obviously be less than or equal to the optimum in the original problem) and then execute classical MBE.

We compare these two approaches. Figure 8.12 reports the lower bounds obtained for different values of parameter z as well as the CPU time required for each execution. The second column shows the best lower bound known for each instance [13]. The fourth, fifth and sixth columns report the lower bound obtained by MO-MBE(z), the time in seconds required to compute it, and the percentage $(\texttt{lb}_{mombe(z)} - \texttt{lb}_{mbe(z)})/\texttt{lb}_{mbe(z)}$. The last three columns

Instance	Best lb known	z	MO-MBE(z)			MBE(z)		
			lb	time	%	lb	time	%
1506	354661	5	244433	0.31	68.92	123174	0.16	34.73
		10	249479	5.04	70.34	138216	1.36	38.97
		15	254513	124.92	71.76	167267	25.74	47.16
1401	459111	5	327152	0.25	71,26	125059	0.12	27.24
		10	333151	5.19	72,56	137060	0.97	29.85
		15	343145	154.66	74,74	165064	21.57	35.95
1403	459268	5	326249	0.3	71,04	121123	0.15	26.37
		10	339265	5.58	73,87	137131	1.22	29.86
		15	340267	156.51	74,09	169144	26.3	36.83
1405	459458	5	322426	0.38	70,18	117170	0.22	25.5
		10	334436	7.25	72,79	150171	1.69	32.68
		15	341452	153.2	74,32	171195	35.58	37.26
1407	459622	5	321475	0.43	69,94	118172	0.28	25.71
		10	342519	6.67	74,52	147205	2.06	32.03
		15	345543	281.7	75,18	175250	41.23	38.13

Figure 8.12: Experimental results on Spot5 instances. Time values in seconds.

reports the same information for MBE(z).

It can be observed that for all instances MO-MBE produces much higher lower bounds than MBE. While this is clearly true if we compare executions with the same value of z, such comparison is not totally fair because MO-MBE has a higher complexity due to the computation of frontier functions. However, if we look at executions with a similar CPU time we still observe a clear dominance of MO-MBE.

8.4 Related Work

The idea of mini-bucket elimination has been made much more general in [82, 84]. In particular, they describe this algorithm applied to valuation algebras. As we have seen in 4.1, our frontier algebra can be considered as a subclass of valuation algebras (recall that the frontier algebra is a c-semiring). Therefore, our MO-MBE can be viewed as an instantiation of those abstract approaches.

A somewhat different approach to approximation is given in [58], which

keeps a careful control of the computation time. The approximation of each bucket is based on a time limited version of the combination operator. The idea is to restrict the available time for the exact combination of functions in each bucket to a given number of seconds. Tuples that have not been computed before the time limit are assigned the top valuation.

It is worth noting that all these approaches are described in an abstract sense. Therefore, non of them are explicitly used to compute the efficient frontier of a multi-objective optimization problem.

8.5 Conclusions

In this chapter we generalize the well-known approximation algorithm MBE to deal with MO-WCSP problems. To the best of our knowledge, MO-MBE is the first approximation algorithm able to compute multi-objective lower bounds. The accuracy parameter z of the mini-bucket technique allows two potential uses of MO-MBE. With high values of z, it can be used to obtain good quality lower bound frontiers of problems that cannot be solved exactly. With low values of z, it can be used as a bounding evaluation function inside any multi-objective branch-and-bound solver to increase its pruning capabilities. We demonstrate the practical potential of MO-MBE in both contexts.

It is important to note that multi-objective branch-and-bound schemes have not been widely used because of the lack of multi-objective approximation algorithms [41, 136]. MO-MBE overcome this issue and allows multi-objective branch-and-bound to be a feasible solving algorithm. Therefore, MO-MBE is the first step toward a deep study of multi-objective branch-and-bound schemes and its associated lower bounding techniques.

Chapter 9

Constraint Propagation

In the previous chapters we have described a number of multi-objective algorithms and have used them to solve multi-objective and mono-objective optimization problems. Now, we propose the use of multi-objective techniques in the solving process of constraint satisfaction problems.

The task in a constraint satisfaction problem (CSP) is to find an assignment satisfying all the constraints. In constaint programming, CSPs are usually solved by search procedures. Essentially, the search process consists of enumerating all possible variable-value combinations, until it finds a solution or proves that none exists. To reduce the exponential number of combinations, search is interleaved with *constraint propagation*. Its goal is to detect and remove domain values that cannot be part of any solution. Constraint propagation is applied at each node of the search tree and, in general, to each constraint independently.

The purpose of this chapter is to describe a novel propagation schema for a set of p constraints of the form $\sum_{f \in \mathcal{F}_j} f < K_j$. The new schema jointly propagates these constraints by means of a multi-objective approximation algorithm. More precisely, we consider the set of additive functions $F_j(\mathcal{X}) = \sum_{f \in \mathcal{F}_j} f$ and compute a lower bound frontier of the corresponding multi-objective problem using MO-MBE (see Chapter 8). If there is no cost vector in the frontier with all its components smaller than the K_j values, it implies

that the current subproblem is inconsistent and the search algorithm can backtrack.

The structure of the Chapter is as follows. Section 9.1 introduces the backtracking algorithm for CSPs and outlines how constraint propagation works. Section 9.2 describes the type of constraints we are going to propagate: additive bounding constraints. Then, we describe two approaches to propagate sets of bounding constraints. Subsection 9.2.1 describes the independent propagation using MBE. Subsection 9.2.2 describes our novel approach: the simultaneos propagation using MO-MBE and highlights the fundamental difference among both approaches. Section 9.3 shows some experimental results. Section 9.4 discusses related work and point out its differences with respect to our approach. Finally, Section 9.5 gives some conclusions.

9.1 Preliminaries

9.1.1 Backtracking

As we have seen in Section 2.3.1, the problem of finding a satisfying assignment with respect to a set of constraints is a constraint satisfaction problem (CSP). Usually, CSPs are solved by a search procedure that systematically enumerates the set of all possible assignments. If the process finds an assignment that satisfies all the constraints then the CSP is consistent. Otherwise, the problem is inconsistent.

The most common search algorithm for CSP problems is *backtracking* [106]. It can be seen as a simplified version of the depth-first branch-and-bound (BB) search for optimization tasks described in Section 3.1.1. Briefly, given a CSP P, the set of all possible assignments, called search space, can be represented as a tree. Each child of a node represents the assignment of one domain value to one additional variable. The path from the root to a given node represents a tuple where only the variables in the path are assigned.

Consider an arbitrary node and let t be its associated assignment. The original CSP problem P conditioned to tuple t (i.e., $P(t)$) is the subproblem rooted at that node. When $P(t)$ does not have any solution, that is, $P(t)$ is inconsistent, the algorithm is in a *dead-end*. In general, a dead-end is detected at nodes where all domain values are rejected as candidates for the assignment of an unassigned variable. When a dead-end is detected, the algorithm backtracks to a previous node. Otherwise, the algorithm continues until reaching a leaf. The assignment associated to the leaf satisfies all the constraints in P. Therefore, P is consistent iff there exists a path from the root to a leaf. Otherwise, P is inconsistent.

9.1.2 Constraint Propagation

It is clear that the efficiency of backtracking depends on the ability to detect dead-ends as soon as possible in the search tree. To that end, constraint programming proposes to solve CSPs by interleaving *constraint propagation* and *search*. Constraint propagation is applied at each node of the search tree. Its goal is to prove the inconsistency of the current subproblem $P(t)$ by proving the inconsistency of some of its constraints.

Definition 9.1.1 *A constraint c is* consistent *iff there exists a tuple t' with $var(c) \subseteq var(t')$ such that t' satisfies c (i.e., $c(t') = true$). Otherwise, c is* inconsistent.

We will refer to constraint propagation as a call to `propagate(C)` where $\mathcal{C}$ is a set of constraints. If `propagate(C)` returns $false$, it means that it has been able to prove that some constraint $c \in \mathcal{C}$ is inconsistent. Formally,

$$\neg\texttt{propagate(C)} \Rightarrow P(t) \text{ is inconsistent}$$

Since constraint propagation is applied in each search node, the consistency of each constraint should be checked efficiently. Let `consistent`(c) refer to the process of checking the consistency of constraint c. When c is

a binary constraint (i.e., c is defined on two variables) $\texttt{consistent}(c)$ is inexpensive. For higher arity constraints this is not necessarily the case. In general, $\texttt{consistent}(c)$ requires time that is exponential in the arity of c. There are two approaches to circumvent this issue:

1. For some specific constraints, it is possible to devise an exact dedicated algorithm that exploits the underlying structure of the constraint. One of the best known specific constraints is the *alldiff* [114] for which a number of dedicated algorithms have been proposed [118, 115, 94]. Such algorithms run in polynomial time and satisfy,

$$\neg\texttt{consistent}(c) \Leftrightarrow c \text{ is inconsistent}$$

2. For some other constraints, it is not possible to check consistency in polynomial time. However, sometimes it is possible to devise an approximated algorithm that may prove inconsistency. A well known example is the propagation of the *knapsack* constraint [75]. Such algorithms satisfy,

$$\neg\texttt{consistent}(c) \Rightarrow c \text{ is inconsistent}$$

With this approach, the propagation of a set of constraints $\mathcal{C}$ can be expressed as the lazy computation of the following expression,

$$\bigwedge_{c \in \mathcal{C}} \texttt{consistent}(c) \tag{9.1}$$

where *lazy computation* means that the propagation process can stop as soon as a constraint is found inconsistent.

The previous approach can be refined in order to, not only prove inconsistency, but also prune inconsistent domain values. This is a well-known idea that in constraint programming is called *filtering*. A filtering algorithm removes inconsistent domain values with respect to a constraint.

Definition 9.1.2 *Given a constraint c and a domain value $a \in D_i$ where $x_i \in var(c)$, a is* inconsistent *with respect to c iff $c(x_i = a)$ is inconsistent.*

Clearly, inconsistent values can be removed without changing the consistency of the current problem. Let `filtering`(c) refer to the process of pruning inconsistent domain values with respect to the constraint c. If it removes all domain values of a variable in $var(c)$, then c is inconsistent.

Filtering algorithms have a very important sinergy: pruning one domain value due to one constraint, may produce the pruning of another domain value due to another constraint, yielding a cascade effect. With this approach, the propagation of a set of constraint $\mathcal{C}$ can be described as the mechanism of calling the consistency algorithm associated with the constraints involving a variable x_i each time the domain of this variable is modified. If the domains of the variables are finite, then it terminates because a domain can be modified only a finite number of times. A naive implementation of this process can be,

function `propagate`($\mathcal{C}$)
    repeat
        for each $c \in \mathcal{C}$ **do** `filtering`(c);
    until domain wiped-out $\vee$ no-change;
    return not domain wiped-out;
endfunction

Although very simple, the previous code allows us to point out three observations. The first observation is that the previous code reduces to Expression 9.1 when `filtering(c)` is replaced by `consistent(c)`. The second observation is that when `propagate` is not able to prove the inconsistency of $P(t)$, the changes made in the domains remain in subsequent subproblems of $P(t)$. The third observation is that the only *communication* between constraints is through value filtering. As a consequence, this approach may not be strong enough for problems with conflicting constraints, as shown in the following example.

Example 9.1.1 *Consider the following two constrains over* 0-1 *variables:*

$$x_1x_2 + x_2x_3 + x_3x_4 \geq 1; \qquad \sum_{i=1}^{4} x_i \leq 1$$

Let us consider that both constraints are associated with a filtering algorithm, which is the strongest form of propagation that we have seen so far. Let us suppose that the propagation process starts with the first constraint. Its associated filtering algorithm iterates over each domain value of each variable in order to detect inconsistent domain values, as follows. First, it considers each domain value of variable x_1*. Domain values* 0 *is consistent with respect to this constraint because there is an assignment with* $x_1 = 0$ *that satisfies it (e.g.* $(x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0)$*). Similarly, domain value* 1 *is consistent because the assignment* $(x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0)$ *satisfies the constraint. It is easy to see that every domain value is consistent with respect to the first constraint. As a result, the filtering algorithm does not remove any domain value and returns true.*

Then, the propagation process will call the filtering algorithm of the second constraint. It is easy to see that when considering any domain value of variable x_i*, the constraint is trivially satisfied when the remaining variables* x_j *with* $j \neq i$ *are assigned to* 0 *(no matter the domain value assigned to* x_i*). As a result, the filtering algorithm does not remove any value and returns true.*

The propagation process returns true, which means that it may be a solution to this problem. The issue is that the independent propagation of each constraint only indicates that there exists an assignment that satisfies each constraint individually. However, this process does not have into account the consistency of each domain value with respect to the two constraints simultaneously. It is easy to see that for satisfying the first constraint we must assign at least two variables to domain value 1*, while for satisfying the second constraint we can assign at most one variable to domain value* 1*. Therefore, no domain value is consistent with both constraints simultaneosly.*

This is a simple example that many solvers would probably deal efficiently with. However, we will show in Section 9.3 that, if constraints are more intricate, standard solvers may perform poorly.

9.2 Propagating Additive Bounding Constraints

In this Chapter we focus on proving inconsistency for an specific constraint, that we call *additive bounding constraint*.

Definition 9.2.1 *An* additive bounding constraint *is a pair* $(\mathcal{F}, K)$*, where* $f \in \mathcal{F}$ *are cost functions and* $K \in \mathbb{N}$ *is a cost value. The scope of the constraint is* $\mathcal{Y} = \cup_{f \in \mathcal{F}} var(f)$*. A tuple* t *such that* $var(t) = \mathcal{Y}$ *satisfies the constraint iff,*

$$\sum_{f \in \mathcal{F}} f(t) < K$$

It is important to note that we do not make any assumption over cost functions $f \in \mathcal{F}$, which makes the concept of additive bounding constraint extremely general.

In recent years, many consistency algorithms for additive bounding constraints have been proposed. For instance, all WCSP local consistency algorithms [88, 32] can be used for filtering.

Another alternative is to use *mini-bucket elimination* MBE (see Section 8.1). Its main disadvantage over local consistencies is that MBE cannot be easily used for filtering domain values [36]. The main advantage is its control parameter z that allows to trade resources for accuracy. For our purposes, MBE has the additional advantage of being extendible to multi-objective optimization (see Chapter 8).

Let P be a CSP with p additive bounding constraints. Consider an arbitrary search node where t is its associated partial assignment. Let $P(t) = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be the problem P conditioned to the current assignment t and let P' be a short-hand for $P(t)$. The set of constraints $\mathcal{C}$ can be divided into two sets: $\{(\mathcal{F}_j, K_j)\}_{j=1}^{p}$ are p additive bounding constraints and $\mathcal{H}$ is the set of

remaining constraints. At this point, standard solvers would start a propagation process using dedicated or approximated algorithms for each type of constraint.

In the following, we consider the propagation process of P' using two schemas:

i. Checking the consistency of *each* bounding constraint independently with MBE.

ii. Checking the consistency of *all* bounding constraints simultaneously using MO-MBE (see Section 8.2).

9.2.1 Propagation using MBE

Checking the consistency of a given bounding constraint can be expressed as,

$$\texttt{lb}_j \geq K_j$$

where $\texttt{lb}_j$ is a lower bound of $\sum_{f \in \mathcal{F}_j} f$. It is easy to see that if the previous expression is satisfied, the j^{th} bounding constraint is inconsistent. Otherwise, the consistency of the constraint remains unknown.

Since the previous consistency checking condition does not filter domain values, it makes sense to propagate P' in two steps. First, we propagate the set $\mathcal{H}$. Then, we sequentially check the consistency of each bounding constraint. Formally, we can see the propagation process as the lazy computation of,

$$\texttt{propagate}(\mathcal{H}) \wedge \bigwedge_{j=1}^{p} \texttt{lb}_j < K_j \tag{9.2}$$

If the previous expression returns *false* the search procedure should backtrack because P' is inconsistent. Otherwise, the search procedure should continue, because P' can be either consistent or inconsistent.

We can compute each $\texttt{lb}_j$ using MBE. Then, the propagation process is a sequence of MBE executions, one for each bounding constraint. Note that it

is important to first propagate $\mathcal{H}$ because its result is the possible reduction of the domains of $\mathcal{X}$ of which MBE takes advantage of.

Example 9.2.1 *Consider a CSP* $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ *with three* $0/1$ *variables and two bounding constraints:* $(F_1, 12)$ *where*

$$F_1 = \{f_1(x_1) = 10x_1, f_2(x_2) = 10x_2, f_3(x_3) = 2x_3\}$$

and $(F_2, 10)$ *where*

$$F_2 = \{h_1(x_1) = 3(1 - x_1), h_2(x_2) = 4(1 - x_2), h_3(x_3) = 8(1 - x_3)\}$$

There are two additional constraints,

$$c_1(x_1, x_2) = x_1 \neq x_2 \qquad c_2(x_2, x_3) = x_2 \vee x_3$$

If we propagate each bounding constraint with MBE setting the control parameter $z = 2$ *we obtain lower bound* 0 *for both constraints. The reason is that the set of functions* $\mathcal{F}_1$ *and* $\mathcal{F}_2$ *only contain unary cost functions, each one mentioning one different variable. Then, the bucket of each variable* x_i *only contains one function that, when* x_i *is projected, it results in a zero-arity function (i.e., a constant). For all buckets, the projected function is* 0*. Then, the lower bound, that is computed as the sum of the set of zero-arity functions, is* 0*.*

The propagation indicates that the problem may have *solution. It is clear that this propagation can be very weak, because it only takes into account the information given by the bounding constraint being propagated.*

The practical effectiveness of MBE can be greatly improved if we add the set of constraints $\mathcal{H}$ to each set $\mathcal{F}_j$. To that end, constraints in $\mathcal{H}$ must be expressed as $0/\infty$–functions.

Example 9.2.2 *Consider the CSP of Example 9.2.1. Now, we are going to propagate each bounding constraint augmented with constraints* c_1 *and* c_2*. In that case, for each bounding constraint, MBE will compute a lower bound*

of $\sum_{f\in\mathcal{F}_j\cup\mathcal{H}} f$*. If we set the control parameter* $z = 2$*, MBE computes lower bounds* 10 *and* 3*, respectively. Find below the trace of each execution (note that in this example, there is no need to break buckets into mini-buckets),*

1st bounding constr.

domain value	g_3	g_2	$g_1()$
1	0	2	10
0	2	10	

2nd bounding constr.

domain value	g_3	g_2	$g_1()$
1	0	4	3
0	0	0	

Note that the lower bounds have increased with respect to the previous propagation. Actually, each lower bound is the optimum assignment of each bounding constraint also considering $\mathcal{H}$*, so MBE is doing a perfect estimation with the information that it receives. However, it is still unable to prove inconsistency. The problem is that it only* knows *part of the information.*

The key of the *possible poor* performance of MBE is that in each execution, MBE searches for one different consistent assignment. Each one satisfies one bounding constraint separately, but may not satisfy all of them simultaneously.

9.2.2 Propagation using MO-MBE

As we have seen, the standard propagation of conflicting bounding constraints may fail. The reason is that it is *easy* to satisfy them independently, but *difficult* to satisfy them simultaneously. Then, the difficulty relies in the conjunction. We propose a more convenient approach where those constraints are simultaneously considered.

The idea is to consider the set of additive bounding constraints $\{(\mathcal{F}_j, K_j)\}_{j=1}^{p}$ as a multi-objective problem with p objective functions $F_1, \ldots, F_p$. The j^{th} objective function is $F_j = \sum_{f\in\mathcal{F}_j} f$. Let $\mathcal{E}$ be the efficient frontier of the multi-objective problem. Then, we compute a lower bound frontier `lbf` of $\mathcal{E}$.

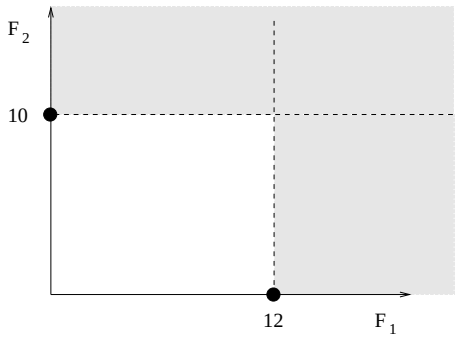

Figure 9.1: Cost vectors from tolerance bounds of bounding constraints $(\mathcal{F}_1, 12)$ and $(\mathcal{F}_2, 10)$.

Let $\vec{u} \in \mathcal{E}$ be an arbitrary vector and let t' be the complete assignment such that $\vec{u} = (F_1(t'), \ldots, F_p(t'))$. Recall that, by definition, $\texttt{lbf} \leq_{mo} \mathcal{E}$. Namely, there exists a vector $\vec{v} \in \texttt{lbf}$ such that $\vec{v} \leq \vec{u}$. If $v_j \geq K_j$ for some j, then $u_j \geq K_j$. As a consequence, t' does not satisfy $(\mathcal{F}_j, K_j)$. Abusing notation, when $v_j \geq K_j$ we say that $\vec{v}$ does not satisfy $(\mathcal{F}_j, K_j)$. If there does not exist any cost vector in `lbf` that satisfies all bounding constraints, neither it exists in $\mathcal{E}$. Namely, there does not exist any complete assignment t' that simultaneosly satisfies all bounding constraints. Therefore, P' is inconsistent.

We illustrate this idea by an example. Consider the CSP problem of Example 9.2.2. It can be transformed into a problem with objective functions $F_1 = \sum_{f \in \mathcal{F}_1} f$ and $F_2 = \sum_{f \in \mathcal{F}_2} f$. The space of solutions can be represented as a 2D space (see Figure 9.1). Any cost vector at the rigth side of the vertical dotted line does not satify the first bounding constraint because its first component will be greater than or equal to $K_1 = 12$. According with the partial order among vectors, all vectors dominated by $(12, 0)$ do not satisfy $(\mathcal{F}_1, 12)$. Similarly, any cost vector above the horizontal dotted line does not satisfy the second bounding constraint because its second component will be greater than or equal to $K_2 = 10$. Namely, all vectors dominated by $(0, 10)$ do not satisfy $(\mathcal{F}_2, 10)$. Only cost vectors situated in the white area simultaneosly satisfies both bounding constraints. In other words, vectors

which are not dominated by either $(12, 0)$ nor $(0, 10)$. It is easy to see that if all vectors in `lbf` are in the dominated area (i.e., grey area in the figure), the CSP problem is inconsistent. However, if only one vector in `lbf` is in the non-dominated are (i.e., white area in the figure), then the CSP may be consistent.

We can easily generalize the previous example to a set of bounding constraints. Since K_j bounds the maximum acceptable cost for the j^{th} objective function F_j, the set of bounds $\{K_j\}_{j=1}^{p}$ can be considered as an upper bound frontier `ubf` of $\mathcal{E}$,

$$\texttt{ubf} = \{(K_1, 0, \ldots, 0), \ldots, (0, \ldots, 0, K_j, 0, \ldots, 0), \ldots, (0, \ldots, 0, K_p)\}$$

According with the partial order among frontiers (see Definition 4.4.2) the expression,

$$\texttt{ubf} \leq_{mo} \texttt{lbf}$$

implies that P' is inconsistent.

Formally, we can see the multi-objective propagation process as the lazy computation of,

$$\texttt{propagate}(\mathcal{H}) \wedge \texttt{ubf} \not\leq_{mo} \texttt{lbf} \tag{9.3}$$

If the previous expression is $false$ the search procedure should backtrack because P' is inconsistent. Otherwise, the search procedure should continue, because P' can be either consistent or not.

We can compute `lbf` using MO-MBE. As in the mono-objective case, the efficiency of MO-MBE can be increased by adding the set of constraints $\mathcal{H}$ to the multi-objective problem. To that end, the set of constraints in $\mathcal{H}$ are expressed as frontier functions. Again, it is important to first propagate $\mathcal{H}$ because it may reduce the domains of $\mathcal{X}$.

Expression 9.3 replaces the sequence of calls to `lb` using MBE in Expression 9.2 by a single call to `lbf` using MO-MBE. This change may seem a minor modification. However, the subjacent algorithm is completely different and the kind of inference performed is much more powerful, as can be seen in the following example.

Example 9.2.3 *Consider the CSP in Example 9.2.1. If we propagate the two bounding constraints simultaneously (augmented with the other constraints) with MO-MBE setting $z = 2$ we obtain a lower bound frontier*

$$\mathtt{lbf} = \{(12, 4), (10, 11)\}$$

Find below the trace of the execution,

domain value	g_3	g_2	$g_1()$
1	$\{(2,0),(0,8)\}$	$\{(2,4),(0,12)\}$	$\{(12,4),(10,11)\}$
0	$\{(2,0)\}$	$\{(10,8)\}$	

The bounds of the two bounding constraints lead to the upper bound frontier $\mathtt{ubf} = \{(12, 0), (0, 10)\}$. *Since* $\mathtt{ubf} \leq_{mo} \mathtt{lbf}$, *the propagation process indicates that the problem* does not *have an assignment satisfying both bounding constraints simultaneously. Therefore, the original CSP problem has no solution.*

The key of the *possible good* performance of multi-objective propagation with respect to MBE is that MO-MBE searches for a unique assignment that satisfies all bounding constraints simultaneously, while the execution of MBE for each bounding constraint searches for one different assignment to satisfy each constraint.

9.3 Experimental Results

We tested our propagation mechanism in the decision (or constraint satisfaction) version of two different domains: *risk-conscious combinatorial auctions* and *scheduling of an EOS* (for more details see Appendix B.1 and B.2, respectively). We compare the performance of four algorithms based on backtracking and using different propagation schemas:

- Each bounding constraint is associated with a dedicated filtering algorithm, called *IloPack*, offered by the constraint solver Ilog Solver 6.1.

- Each bounding constraints (augmented with the other different constraints) is associated with an approximated filtering algorithm that enforces FDAC [88].
- The set of constraints different from the bounding constraints are jointly propagated with an approximated algorithm that enforces arc-consistency on them. The consistency of each bounding contraint is proved with MBE setting $z = 2$ (namely, the first approach described in Section 9.2.1).
- Similar to the previous one, but the two calls to MBE are replaced to one call to MO-MBE setting $z = 2$ (namely, the second approach described in Section 9.2.2).

We use Ilog Solver 6.1 as solver engine for the first approach, and Toolbar in the other three approaches. For comparison, we always report cpu time.

9.3.1 Combinatorial Auctions

Figure 9.2 reports the results obtained for the risk conscious auction instances with 20 and 50 goods, respectively. The time limit is 300 seconds.

It can be observed that problems become harder as the number of bids increases. Regarding the algorithms, it is clear that MO-MBE propagation always outperforms the other three approaches. For instances with 20 goods, it is about 6 times faster than its competitors. With 50 goods the gain is still larger (up to 10 times faster).

9.3.2 Scheduling of an EOS

The time limit for this experiment is 600 seconds. Since we could not solve complete instances, we considered subinstances as follows: $X_{\geq k}$ denotes instance X where photographs whose penalty is less than k have been eliminated. Figure 9.3 reports the results for instance $1506_{\geq 1000}$. Since we observed that the behavior of other subinstances (i.e., $1401_{\geq 1000}$, $1403_{\geq 1000}$,

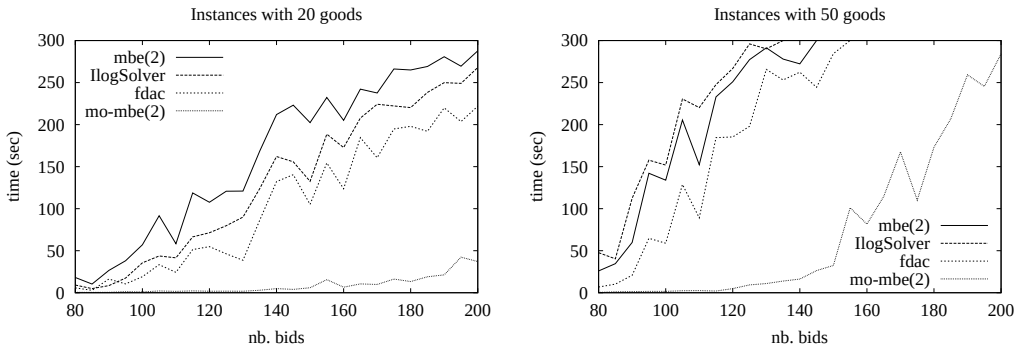

Figure 9.2: Experimental results on risk conscious combinatorial auctions for 20 and 50 goods, respectively. Path distribution. Risk probabilities ranging from 0.0 to 0.3. Average time on 25 instances for each parameter configuration. Time limit 300 seconds.

$1405_{\geq 1000}$, and $1407_{\geq 1000}$) was very similar, we do not report their results. Each plot reports results for a fixed value of P and varying the value of S. Note the logarithmic scale.

IlogSolver always performs very poorly and only solves instances with $S \leq 4$. Thus, we omit it from the plot.

Considering MBE and MO-MBE, we observe the following pattern that is best exemplified in the $P = 450000$ plot (Figure 9.3 top left). For high values of S, MBE is more efficient than MO-MBE. The reason is that the memory constraint is very easy to satisfy, which makes it practically irrelevant. MBE already captures the difficulty of the problem, which is mono-objective in nature. Thus, the higher overhead of MO-MBE is wasted. As the value of S decreases, the situation changes. Both bounding constraints become difficult to satisfy simultaneously. Propagating with mono-objective MBE fails in detecting inconsistency because it is easy to satisfy each constraint if the other one is disregarded, but it is difficult to satisfy the two of them simultaneously. Only the bi-objective nature of MO-MBE can capture such difficulty. As a result, MBE cannot solve the problems, while MO-MBE solves them in

a few seconds. If S decreases even further, the memory constraint becomes clearly unsatisfiable in conjunction with the penalty constraint. MO-MBE propagation detects it easily but MBE propagation does not. Only for the lowest values of S, when the constraint is unsatisfiable independently of other constraints, MBE detects it efficiently. The algorithm that enforces FDAC behaves similarly to MBE because it also considers the two bounding constraints separately. However, it provides a much better average performance.

Observing the plots in decreasing order of P, we observe that problems become harder as the penalty bounding constraint becomes tighter and harder to satisfy. As before, there is a range of S for which the instances are most difficult. This difficulty peak shifts towards the right as P decreases. For MO-MBE propagation, the range is narrower than for MBE and FDAC, but it also fails to solve some instances within the time limit of 600 seconds.

The $P = 250000$ case requires further discussion: the plot only shows the left-hand side of the difficulty peak, where the tight memory constraint *helps* MO-MBE to prove unsatisfiability almost instantly whilst MBE and FDAC cannot. For large values of S the constraint becomes trivial and irrelevant. Then the problem difficulty is given only by the penalty constraint and the three algorithms fail in solving it.

9.4 Related work

The idea of using the conjunction of two or more constraints during propagation, rather than using them one-by-one, is not new. For instance, path-consistency, path-inverse consistency and neighborhood inverse consistency [33] use this idea at different levels of sophistication. However, all these works assume binary problems and cannot be efficiently extended to higher arity constraints such as bounding constraints. The work of [141] is also related to ours. However, it is restricted to problems with so-called *knapsack constraints*, which are a special case of pairs of additive bounding constraints that share unary cost functions (namely, linear constraints of the

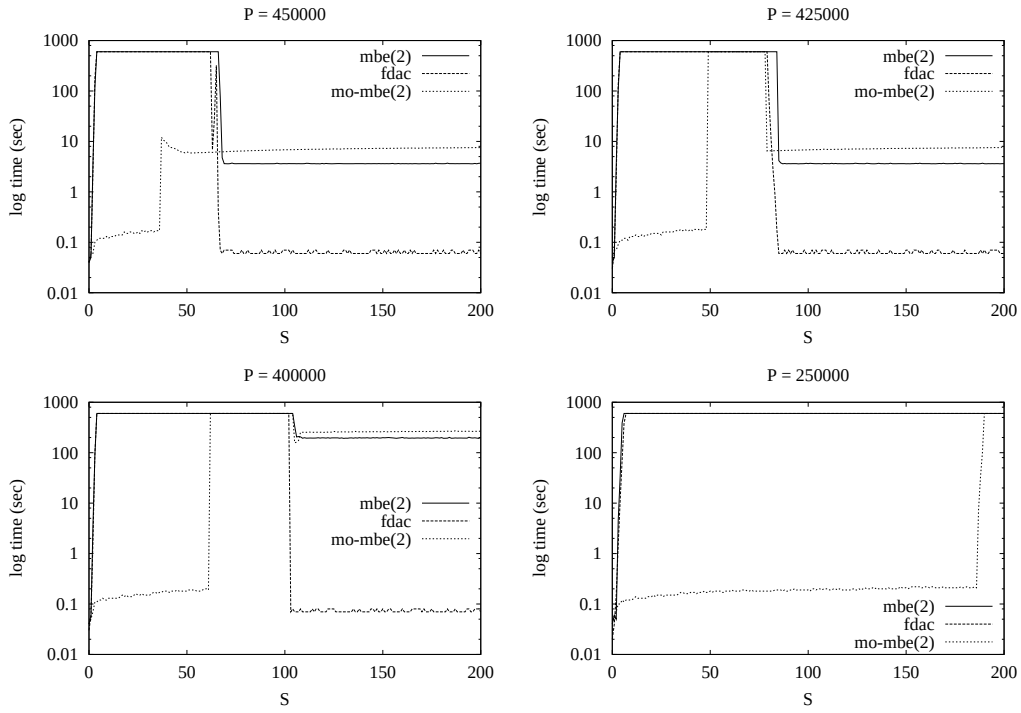

Figure 9.3: Experimental results on $1506_{\geq 1000}$ spot5 instance. Time limit 600 seconds. Note the logarithmic scale.

form $L \leq AX \leq U$). A little bit more general is the work of [131], which applies to pairs of constraints of the form,

$$\sum_{i=1}^{n} w_i x_i \leq U \quad \wedge \quad \sum_{i=1}^{n} p_i x_i > U$$

Our notion of additive bounding constraint includes these and many other cases and allow us to take into account any number of bounding constraints. Besides, it can be easily extended to more sophisticated bounding constraints expressable in terms of semirings [17]. Overmore, our algorithmic approach using multi-objective optimization techniques is radically different.

9.5 Conclusions

Additive bounding constraints, $\sum_{f \in \mathcal{F}} f(X) < K$, are used to bound the tolerance under certain undesirable feature in problem solutions. The propagation in problems involving conflicting bounding constraints is a difficult task for standard solvers. Typically, they propagate constraints one by one. When it is easy to satisfy bounding constraints independently, but difficult to satisfy them simultaneously, this approach clearly fails. In this Chaper we have proposed a novel approach inspired in multi-objective optimization. We propagate the additive bounding constraints simultaneously with multi-objective mini-bucket elimination (MO-MBE). The output is a multi-objective lower bound frontier that can be used to detect the inconsistency of the problem. Our experiments on two domains inspired in real-world problems show that propagation of additive bounding constraints using MO-MBE is clearly superior than previous approaches.

The high overhead of multi-objective propagation may render it useless in problems with many bounding constraints. In that case, it may be useful to detect automatically pairs of conflicting constraints and apply MO-MBE to these pairs independently. Moreover, the experiments indicated that lose bounding constraints cause overhead but are of no use to our approach, so they should be detected and discarded in the propagation process. The development of this idea is part of our future work. A major drawback of MO-MBE propagation is that it cannot detect and prune unfeasible values. We want to overcome this problem using the ideas of [36].

Chapter 10

Conclusions and Future Work

Many important real world optimization problems involve multiple objective functions that should be simultaneously optimized. Multi-objective optimization is characterized by a set of uncomparable solutions, instead of a unique, perfect solution. It is obvious that solving this type of problems is not trivial. Therefore, the development of techniques to efficiently solve them is of clear practical importance.

Graphical models is a common representation framework to model a wide spectrum of combinatorial problems, such as mono-objective optimization or counting problems. Research during the last three decades has produced a collection of general algorithms to efficiently solve them. The unifying view provides a bridge to transfer specialized techniques from one type of graphical model to another.

In this Thesis we have studied multi-objective optimization problems under the graphical model framework. We have extended many techniques developed in the graphical model context to multi-objective optimization. All our research has been motivated under a general-purpose perspective, without assuming any domain knowledge. For this reason, we believe that our constributions can be effective in a broad spectrum of domains.

Our work has some recognized limitations. For instance, our empirical evaluation only considers bi-objective optimization problems. Although real-

world problems will generally consider a small number of objectives (i.e., no more than three), the increase in the number of objectives will presumably decrease the performance of our algorithms. Moreover, all our benchmarks have some source of randomness. It should be clear that results obtained on random problems need not extrapolate to every particular domain. Finally, we have disregarded local search methods, an important line of research which has been very fruitful and widely studied in multi-objective optimization.

10.1 Conclusions

The main conclusions of our work are:

1. Many multi-objective optimization problems can be described in graphical models terms. For the first time, we develop a valid formalization and prove that it satisfies all the axioms required by the semiring CSP (SCSP) framework for graphical models.

2. The formalization of multi-objective optimization problems as SCSP problems gives us the main elements to naturally extend some algorithms described in the mono-objective optimization context. We have presented new search and inference general purpose algorithms able to compute either the efficient frontier or a multi-objective approximation of a given multi-objective optimization problem. The clear parallelism between multi-objective and mono-objective algorithms allows us to better understand their algorithmic structure. As we have seen, previous existing work describes such elements in a not so intuitive way.

3. The new proposed algorithms behave well in multi-objective optimization problems with different structural characteristics, as we have empirically demonstrated. The following table indicates the most efficient algorithm (in general) for each benchmark taking into account the structural characteristics of the instances.

	Max-SAT-ONE			BMWVC[1]		CA[2]	EOS[3]
	$w^* < 24$	$w^* \geq 24$		$w^* < 24$	$w^* \geq 24$		
		$n = 50$	$n \geq 100$				
MO-BB$_{\texttt{icf}}$		$\surd$					
MO-BB$_{fdac}$					$\surd$		
(S)MO-RDS							$\surd$
MO-BE	$\surd$			$\surd$			
MO-BB$_{mombe}$						$\surd$	
MO-MBE			$\surd$				

It is well-known that there does not exist the *best* algorithm in terms of efficiency. On the contrary, there exist algorithms that are suitable for problems with relatively small induced width, others are convenient for problems with relatively small bandwidth, etc. Our *small* sample of benchmark is a clear example. In general, the structure of each problem determines whether one algorithm should be in principle more suitable than another. Therefore, it is important to have a collection of algorithms that performs well in different situations.

4. Our work supports the importance of developing generic algorithms which, in general, are easier to develop and maintain, and can be a starting point for the development of specialized techniques. That is why all our work has been developed under a general purpose motivation. Moreover, we believe that the new algorithms can be specialized to particular domains without jeopardizing their performance. On the contrary, the specialization should lead to more efficient algorithms.

[1]Biobjective minimum weighted vertex cover
[2]Risk-conscious combinatorial auctions
[3]Scheduling of an Earth Observation Satellite

5. Finally, multi-objective optimization techniques can be useful to solve pure satisfaction and mono-objective optimization problems. This approach may be counterintuitive at first sight because multi-objective optimization is in general more difficult than mono-objective optimization. However, we have shown that the multi-objective perspective may bring to light desirable structural properties that renders multi-objective optimization algorihtms more efficient than any decision or mono-objective technique.

10.2 Future work

This work raises a number of issues that deserve further research. We have identifyed the following related to *AND/OR search*, *hybrid approaches*, *search heuristics*, and *local consistency techniques*.

1. As we have seen in Section 3.1.1, there are two main search approaches in mono-objective optimization: OR search and AND/OR search. The main advantage of the latter is that it takes into account the independences among variables. As a result, the search space traversed by AND/OR search is smaller than the one traversed by OR search. The main consequence is the exponential reduction of the search time with respect to the traditional OR search. In our future work, we want to extend AND/OR search to multi-objective optimization. Thanks to the algebraic formalization of multi-objective optimization problems described in Chapter 4, the extension should be straightforward. Moreover, we also want to extend the catching schemas proposed for AND/OR search to the multi-objective context. The result will be a parameterized algorithm able to trade time and space. Given the space limitations in real life problems, exact solving schemes that can trade space for time are of clear importance.

2. Hybrid approaches combine in a single method different solving tech-

niques. The goal of the combination is to retain the qualities of each approach while minimizing some of their drawbacks. In mono-objective optimization, the combination of search and bucket elimination [87] has been shown to be very effective in several domains. We want to investigate the efficiency of this hybrid approach in the multi-objective optimization context. It is our belief that it may be a suitable combination, but an empirical evaluation is needed.

3. The search for solutions is generally guided by two heuristics defining the *search strategy* (i.e., variable [53, 147, 62] and domain value [53, 147, 146] selection heuristics). The goal of a search strategy is to quickly guide search towards either good solutions or dead-ends. It is known that in mono-objective optimization a good search strategy has a great impact on the solving time. In our future work, we want to study the impact of introducing different search strategies into the multi-objective branch-and-bound algorithm.

4. Approximation algorithms are of clear interest either when an optimization problem is too difficult to be solved exactly, or when combined with search. We have investigated the efficiency of multi-objective mini-bucket elimination in both contexts. In our future work, we want to explore the extension of local consistency techniques to multi-objective optimization. The main issue is the definition of an inverse of the combination operator over frontiers able to extract as much information as possible from the network. In a preliminary study, we define this inverse operator as the pointwise difference among an arbitrary frontier and a singleton frontier. However, we want to further investigate whether it is possible to define a stronger inverse operator.

Bibliography

[1] *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*. AAAI Press, 2007.

[2] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[3] K. Apt. *Principles of Constraint Programming*. Cambridge University Press, Cambridge, 2003.

[4] P. Armand. Finding all maximal efficient faces in multiobjective linear programming. *Mathematical Programming*, 61:357–375, 1993.

[5] Y. Asahiro, K. Iwama, and E. Miyano. *Random Generation of Test Instanzes with Controlled Attributes*, pages 377–394. Volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* [70], 1996.

[6] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #sat and bayesian inference. In *FOCS*, pages 340–351. IEEE Computer Society, 2003.

[7] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Value elimination: Bayesian interence via backtracking search. In Christopher Meek and Uffe Kjærulff, editors, *UAI*, pages 20–28. Morgan Kaufmann, 2003.

[8] Roberto J. Bayardo and Daniel P. Miranker. A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In *Proc. of the 13thAAAI*, pages 298–304, Portland, OR, 1996.

[9] Matthew J. Beal, Nebojsa Jojic, and Hagai Attias. A graphical model for audiovisual object tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(7):828–836, 2003.

[10] F. A. Behringer. Lexicographic quasiconcave multiobjective programming. *Zeitschrift für Operations Research*, 21:103–116, 1977.

[11] Frédéric Benhamou, editor. *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, volume 4204 of *Lecture Notes in Computer Science*. Springer, 2006.

[12] E. Bensana, M. Lemaitre, and G. Verfaillie. Earth observation satellite management. *Constraints*, 4(3):293–299, 1999.

[13] E. Bensana, G. Verfaillie, J. Agnse, N. Bataille, and D. Blumstein. Exact and inexact methods for the daily management of an earth observation satellite. In *In Proc. 4 th Intl. Symposium on Space Mission Operations and Ground Data Systems, Munich, Germany, 1996.*, 1996.

[14] H.P.Van Benthem. Graph: Generating radio link frequency assignment problems heuristically. Master's thesis, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands., 1995.

[15] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972.

[16] D. Bertsekas. *Dynamic Programming*. Prentice Hall, Englewook Cliffs, 1987.

[17] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints*, 4:199–240, 1999.

[18] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, March 1997.

[19] S. Bistarelli, U. Montanari, and F. Rossi. Sclp semantics for (multi-criteria) shortest path problems. In *CP-AI-OR-99*, 1999.

[20] Hans L. Bodlaender, John R. Gilbert, Ton Kloks, and Hjálmtyr Hafsteinsson. Approximating treewidth, pathwidth, and minimum elimination tree height. In Gunther Schmidt and Rudolf Berghammer, editors, *WG*, volume 570 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1991.

[21] Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for max-sat. *Artif. Intell.*, 171(8-9):606–618, 2007.

[22] Sylvain Bouveret and Michel Lemaître. New constraint programming approaches for the computation of leximin-optimal solutions in constraint networks. In Veloso [144], pages 62–67.

[23] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Constraints*, 4:79–89, 1999.

[24] V. Chankong and Y. Y. Haimes. *Multiobjective Decision Making: Theory and Methodology*. Elsevier Science Publishing Co., New York, 1983.

[25] Carlos A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowl. Inf. Syst.*, 1(3):129–156, 1999.

[26] Stephen Cook. The complexity of theorem proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[27] M. Cooper. High-order consistency in valued constraint satisfaction. *Constraints*, 10:283–305, 2005.

[28] M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2):199–227, 2004.

[29] R.G. Cowell, A.P. Dawid, S. L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, 1999.

[30] Adnan Darwiche. Recursive conditioning. *Artif. Intell.*, 126(1-2):5–41, 2001.

[31] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 3, 1960.

[32] S. de Givry, F. Heras, J. Larrosa, and M. Zytnicki. Existential arc consistency: getting closer to full arc consistency in weighted csps. In *Proc. of the 19th IJCAI*, Edinburgh, U.K., August 2005.

[33] R. Debruyne and C. Bessière. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, 2001.

[34] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[35] R. Dechter. *Constraint Processing*. Morgan Kaufmann, San Francisco, 2003.

[36] R. Dechter, K. Kask, and J. Larrosa. A general scheme for multiple lower bound computation in constraint optimization. In *CP-2001*, pages 346–360, 2001.

[37] R. Dechter and R. Mateescu. And/or search spaces for graphical models. *Artif. Intell.*, 171(2-3):73–106, 2007.

[38] R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM*, 50(2):107–153, March 2003.

[39] M. Ehrgott. A characterization of lexicographic maxordering solutions. In *Methods of Multicriteria Decision Theory: Proceedings of the 6th Workshop of the DGOR Working-Group Multicriteria Optimization and Decision Theory.*, pages 193–202, 1996.

[40] M. Ehrgott. Discrete decision problmes, multiple criteria optimization classes and lexicographica max-ordering. *Lecture Notes in Economics and Mathematical Systems*, 465:31–44, 1998.

[41] M. Ehrgott and X. Gandibleux. *Multiple Criteria Optimization. State of the Art. Annotated Bibliographic Surveys.* Kluwer Academic Publishers, 2002.

[42] M. Ehrgott, J. Puerto, and A.M. Rodríguez-Chía. Primal–dual simplex method for multiobjective linear programming. *Journal of Optimization Theory and Applications*, 134:483–497, 2007.

[43] J. P. Evans and R. E. Steuer. A revised simplex method for linear multiple objective programs. *Mathematical Programming*, 5:375–377, 1973.

[44] Torsten Fahle. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In Rolf H. Möhring and Rajeev Raman, editors, *ESA*, volume 2461 of *Lecture Notes in Computer Science*, pages 485–498. Springer, 2002.

[45] F. Joel Ferguson and Tracy Larrabee. Test pattern generation for realistic bridge faults in cmos ics. In *ITC*, pages 492–499. IEEE Computer Society, 1991.

[46] M. Fishelson and D. Geiger. Exact genetic linkage computations for general pedigrees. *Bioinformatics*, 2002.

[47] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. In Stephanie Forrest, editor, *ICGA*, pages 416–423. Morgan Kaufmann, 1993.

[48] Carlos M. Fonseca and Peter J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

[49] William T. Freeman, Egon C. Pasztor, and Owen T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.

[50] E. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29:24–32, March 1982.

[51] E. C. Freuder. Synthesizing constraint expressions. *Communications ACM*, 21(11):958–966, 1978.

[52] E.C. Freuder and M.J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *IJCAI-85*, pages 1076–1078, 1985.

[53] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, December 1992.

[54] Marco Gavanelli. An algorithm for multi-criteria optimization in csps. In Frank van Harmelen, editor, *ECAI*, pages 136–140. IOS Press, 2002.

[55] Michel Gendreau, Patrick Soriano, and L. Salvail. Solving the maximum clique problem using a tabu search approach. *Annal of Operations Research*, 41(4):385–403, 1993.

[56] A. M. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22:618–630, 1968.

[57] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural csp decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.

[58] Rolf Haenni. Ordered valuation algebras: a generic framework for approximating inference. *Int. J. Approx. Reasoning*, 37(1):1–41, 2004.

[59] S. Harikumar and S. Kumar. Iterative deepening multiobjective A*. *Information Processing Letters*, 58:11–15, 1996.

[60] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Trans. Syst. Sci. Cybern. SSC-4*, pages 100–107, 1968.

[61] P.E. Hart, N.J. Nilsson, and B. Raphael. Correction to 'a formal basis for the heuristic determination of minimum cost paths'. In *SIGART Newsletter*, volume 37, pages 28–29, 1972.

[62] Federico Heras and Javier Larrosa. Intelligent variable orderings and re-orderings in dac-based solvers for wcsp. *J. Heuristics*, 12(4-5):287–306, 2006.

[63] A. Holland. *Risk Management for Combinatorial Auctions*. PhD thesis, Dept. of Computer Science, UCC, Ireland., 2005.

[64] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *International Conference on Evolutionary Computation*, pages 82–87, 1994.

[65] R. Howard and J. Matheson. Influence diagrams. In *Readings on the Principles and Applications of Decision Analysis*, pages 719–762, Menlo Park, CA, USA, 1984. Strategic Decisions Group.

[66] F. Le Huédé, M. Grabisch, C. Labreuche, and P. Savéant. Mcs – a new algorithm for multicriteria optimisation in constraint programming. *Annals of Operation Research*, 147:143–174, 2006.

[67] H. Isermann. Proper efficiency and the linear vector maximum problem. *Operations Research*, 22:189–191, 1974.

[68] H. Isermann. The enumeration of the set of all efficient solutions for a linear multiple objective program. *Operational Research Quaterly*, 28(3):711–725, 1977.

[69] S. Greco J. Figueira and M. Ehrgott. *Multiple Criteria Decision Analysis. State of the Art. Surveys*. Springer Science+Business Media, Inc., 2005.

[70] D. S. Johnson and M. Trick. *Second DIMACS implementation challenge: cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[71] M.I. Jordan. *Learning in Graphical models*. MIT Press, 1998.

[72] Ulrich Junker. Preference-based search and multi-criteria optimization. *Annals of Operation Research*, 130:75–115, 2004.

[73] Richard M. Karp. *Reducibility Among Combinatorial Problems*, pages 85–103. New York: Plenum. R. E. Miller and J. W. Thatcher, 1972.

[74] K. Kask and R. Dechter. A general scheme for automatic generation of search heuristics from specification dependencies. *Artificial Intelligence*, 129:91–131, 2001.

[75] Irit Katriel, Meinolf Sellmann, Eli Upfal, and Pascal Van Hentenryck. Propagating knapsack constraints in sublinear time. In *AAAI* [1], pages 231–236.

[76] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley and Sons, 1976.

[77] T. P. Kirkman. On a problem in combinatorics. *Cambridge Dublin Math. J. 2*, pages 191–204, 1847.

[78] M.Pearson K.Leuton-Brown and Y.Shoham. Towards a universal test suite for combinatorial auction algorithms. *ACM E-Commerce*, pages 66–76, 2000.

[79] Joshua D. Knowles and David Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[80] J. Kohlas. *Information Algebras: Generic Structures for Inference*. Springer-Verlag, 2003.

[81] J. Kohlas and P. Shenoy. Computation in valuation algebras. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 5, pages 5–40, 2000.

[82] J. Kohlas and N. Wilson. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artif. Intell.*, 172(11):1360–1399, 2008.

[83] R. Korf. Depth-first iterative-deepening: An optimal admissable tree search. *Artificial Intelligence*, 27(3):97–109, 1985.

[84] L-Chang and A.K. Mackworth. Generalized constraint-based inference. Tech. Rep. TR-2005-10, 2005.

[85] Tracy Larrabee. Test pattern generation using boolean satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 11(1):4–15, 1992.

[86] J. Larrosa. On the time complexity of bucket elimination algorithms. Technical report, University of California at Irvine, 2001.

[87] J. Larrosa and R. Dechter. Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. *Constraints*, 8(3):303–326, 2003.

[88] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted csp. In *Proc. of the 18th IJCAI*, Acapulco, Mexico, August 2003.

[89] J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc-consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.

[90] Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient max-sat solving. *Artif. Intell.*, 172(2-3):204–233, 2008.

[91] Javier Larrosa and Emma Rollon. Adaptive consistency with capacity constraints. In *Workshop on Modelling and Solving Problems with Constraints. ECAI'04*, 2004.

[92] Javier Larrosa and Emma Rollon. Bucket elimination with capacity constraints. In *6th Workshop on Preferences and Soft Constraints. CP'04*, 2004.

[93] S.L. Lauritzen and N.A. Sheehan. Graphical models for genetic analysis. *Stat. Sci.*, 18:489–514, 2003.

[94] M. Leconte. A bounds-based reduction scheme for constraints of difference. In *Constraints-96, Second International Workshop on Constraint-based Reasoning*, 1996.

[95] X.Y. Li. *Optimization Algorithms for the Minimum-Cost Satisfiability Problem*. PhD thesis, North Carolina State University, 2004.

[96] A. Mackworth. Consistency in networks of constraints. *Artificial Intelligence*, 8, 1977.

[97] Lawrence Mandow and José-Luis Pérez de-la Cruz. A new approach to multiobjective A* search. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 218–223. Professional Book Center, 2005.

[98] Radu Marinescu and Rina Dechter. Memory intensive branch-and-bound search for graphical models. In *AAAI*. AAAI Press, 2006.

[99] Radu Marinescu and Rina Dechter. Best-first and/or search for graphical models. In *AAAI* [1], pages 1171–1176.

[100] Robert Mateescu and Rina Dechter. The relationship between and/or search and variable elimination. In *UAI*, pages 380–387. AUAI Press, 2005.

[101] Robert Mateescu and Rina Dechter. A comparison of time-space schemes for graphical models. In Veloso [144], pages 2346–2352.

[102] Robert J. McEliece, David J. C. MacKay, and Jung-Fu Cheng. Turbo decoding as an instance of pearl's "belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152, 1998.

[103] Pedro Meseguer and Martí Sánchez. Specializing russian doll search. In Toby Walsh, editor, *CP*, volume 2239 of *Lecture Notes in Computer Science*, pages 464–478. Springer, 2001.

[104] U. Montanari. Networks of constraint fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[105] H. Moulin. *Axioms of Cooperative Decision Making*. Cambridge University Press, Cambridge, 1988.

[106] N. Nilsson. *Principles of Artificial Intelligence*. Springer-Verlag, Berlin Heidelberg New York, 1982.

[107] P.P. Chakrabarti P. Dasgupta and S. C. DeSarkar. Multiobjective heuristic search in and/or graphs. *Journal of Algorithms*, 20:282–311, 1996.

[108] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[109] James D. Park. Using weighted max-sat engines to solve mpe. In *Proc. of the 18th AAAI*, pages 682–687, Edmonton, Alberta, Canada, 2002.

[110] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison–Wesley, Reading, Massachusets, 1985.

[111] J. Pearl. *Probabilistic Reasoning in Intelligent Systems. Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[112] Cédric Pralet, Thomas Schiex, and Gérard Verfaillie. Decomposition of multi-operator queries on semiring-based graphical models. In Benhamou [11], pages 437–452.

[113] Cédric Pralet, Gérard Verfaillie, and Thomas Schiex. Decision with uncertainties, feasibilities, and utilities: Towards a unified algebraic framework. In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *ECAI*, pages 427–431. IOS Press, 2006.

[114] J.C. Puget. Global constraints and filtering algorithms. In *Constraints and Integer Programming Combined*. Kluwer, M. Milano editor, 2003.

[115] Jean-Francois Puget. A fast algorithm for the bound consistency of alldiff constraints. In *AAAI/IAAI*, pages 359–366, 1998.

[116] L.R. Rabiner. A tutorial in hidden markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2), 1989.

[117] A. Raval, Zoubin Ghahramani, and David L. Wild. A bayesian network model for protein fold and remote homologue recognition. *Bioinformatics*, 18(6):788–801, 2002.

[118] J.C. Regin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th AAAI*, pages 362–367, 1994.

[119] E. Rollon and J. Larrosa. Depth-first mini-bucket elimination. In *Proc. of the 11th CP*, pages 563–577, Sitges (Spain), 2005. LNCS 3709. Springer-Verlag.

[120] E. Rollon and J. Larrosa. Bucket elimination for multiobjective optimization problems. *Journal of Heuristics*, 12(4-5):307–328, 2006.

[121] E. Rollon and J. Larrosa. Mini-bucket elimination with bucket propagation. In Benhamou [11], pages 484–498.

[122] E. Rollon and J. Larrosa. Multi-objective propagation in constraint programming. In *Proc. of the 17th European Conference on Artificial Intelligence, ECAI'06*, 2006.

[123] E. Rollon and J. Larrosa. Multi-objective russian doll search. In *AAAI* [1], pages 249–254.

[124] E. Rollon and J. Larrosa. Constraint optimization techniques for multiobjective branch-and-bound search. *Lecture Notes in Economics and Mathematical Systems*, In press, 2008.

[125] Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976.

[126] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming.*, chapter 9. Elsevier, 2006.

[127] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *IJCAI-99*, pages 542–547, 1999.

[128] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In John J. Grefenstette, editor, *ICGA*, pages 93–100. Lawrence Erlbaum Associates, 1985.

[129] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *IJCAI-95*, pages 631–637, Montréal, Canada, August 1995.

[130] E. Segal, M. Shapira, A. Regev, D. Peer, D. Botstein, D. Koller, and N. Friedman. Module networks: Identifying regulatory modules and their condition-specific regulators from gene expression data. *Nature Genetics*, 34(2):166–176, 2003.

[131] Meinolf Sellmann. Approximated consistency for knapsack constraints. In *CP 2003*, pages 679–693. LNCS 2833. Springer Verlag, 2003.

[132] Bart Selman, David G. Mitchell, and Hector J. Levesque. Generating hard satisfiability problems. *Artif. Intell.*, 81(1-2):17–29, 1996.

[133] G. R. Shafer and P.P. Shenoy. Probability propagation. *Anals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.

[134] P. Shafer. An axiomatic study of computation in hypertrees. Working paper 232, School of Business, University of Kansas, 1991.

[135] P. Shenoy. Axioms for dynamic programming. In *Computational Learning and Probabilistic Reasoning*, pages 259–275, 1996.

[136] Francis Sourd, Olivier Spanjaard, and Patrice Perny. Multi-objective branch and bound. application to the bi-objective spanning tree problem. In *7th International Conference in Multi-Objective Programming and Goal Programming*, 2006.

[137] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[138] R.E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. John Wiley and Sons, New York, 1985.

[139] B. S. Stewart and C. C. White. Multiobjective A*. *Journal of the ACM*, 38(4):775–814, 1991.

[140] S. Tiourine, C.Hurkens, and J.Lenstra. A review of algorithmic approaches to frequency assigment problems. Technical report, Rapport technique n. 3507.00/W.E.2.5.4. Eindhoven University of Technology. CERT, 1995.

[141] Michael Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operation Research*, 118(118):73–84, 2003.

[142] Zhuowen Tu, Xiangrong Chen, Alan L. Yuille, and Song Chun Zhu. Image parsing: Unifying segmentation, detection, and recognition. *International Journal of Computer Vision*, 63(2):113–140, 2005.

[143] E. L. Ulungu and J. Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3:83–104, 1994.

[144] Manuela M. Veloso, editor. *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2007.

[145] G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search. In *AAAI-96*, pages 181–187, Portland, OR, 1996.

[146] R. Wallace. Directed arc consistency preprocessing as a strategy for maximal constraint satisfaction. In M. Meyer, editor, *ECAI94 Workshop on Constraint Processing*, pages 69–77, 1994.

[147] Richard J. Wallace and Eugene C. Freuder. Conjunctive width heuristics for maximal constraint satisfaction. In *AAAI*, pages 762–768, 1993.

[148] J. Whittaker. *Graphical Models in Applied Multivariate Statistics*. John Wiley and Sons, 1990.

[149] Ramin Zabih. Some applications of graph bandwidth to constraint satisfaction problems. In *AAAI*, pages 46–51, 1990.

[150] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Trans. Evolutionary Computation*, 3(4):257–271, 1999.

Appendix A

Engineering Mini-Bucket Elimination

As we have seen in Chapter 8, Mini-bucket Elimination (MBE) is one of the most popular bounding techniques. It is arguably one of the best-known general approximation algorithms for mono-objective optimization problems and it has shown to be effective in a variety of *graphical models*. The time and space complexity of MBE is exponential in its control parameter z and it is important to note that, with current computers, it is the space, rather than the time, that prohibits the execution of the algorithm beyond certain values of z.

The purpose of this Chapter is to improve the practical applicability of MBE. To that end, we propose two complimentary methods. Given a value of the control parameter z, the first method decreases the space demands and obtains the same lower bound as the original MBE. The second one increases the lower bound and maintains the same space demans as the original MBE.

The Chapter is divided into two sections, each one devoted to one new method.

A.1 Improving MBE memory usage

In this Section we show how to decrease the space demands of MBE. The new method is based on the concept of *computation tree* (CT). A CT provides a graphical view of the MBE execution and can be computed as a pre-process. It is somewhat similar to the tree-decomposition in decomposition methods [38], where the first step is to build the tree-decomposition and the second step is to solve the problem. Our first contribution is a set of local transformations to the CT with which a more rational use of memory is achieved. They include: *i*) *branch re-arrangement* (nodes are moved upwards along a branch which means that the elimination of a variable is anticipated) and, *ii*) *vertical tree compaction* (adjacent nodes are joined which means that a sequence of operations is performed in a single step).

The second contribution is the exploitation of *memory deallocation* of intermediate functions when they become redundant. By construction of CT, MBE can be seen as a top-down traversal of the CT. The order of the traversal is imposed by the order in which variables are eliminated. We make the observation that any top-down traversal of the CT would produce the same outcome. Then, we propose to traverse the CT in a *depth-first* manner in order to decrease the number of intermediate functions that must be simultaneously stored. We show that with a depth-first traversal of the CT, the order of children has an impact in the space complexity which provides an additional source of improvement. We also discuss the benefits of *horizontal node compaction*. It is important to note that none of these transformations risk the accuracy of the algorithm.

The new algorithm that incorporates all these techniques is called depth-first mini-bucket elimination dfMBE. Our experiments show in a number of domains that dfMBE may provide important space savings. The main consequence is that in a given computer (namely, for a fixed amount of memory), dfMBE(z) can be executed with a higher value of z than MBE(z) which, in turn, may yield better lower bounds.

A.1.1 Preliminaries

The first phase of MBE (as well as BE) can be seen as an algebraic expression that combines sums and variable eliminations. For instance, consider the following set of cost functions,

$$\mathcal{F} = \{f_1(x_6, x_5, x_4), f_2(x_6, x_5, x_3), f_3(x_5, x_3, x_2), f_4(x_6, x_4, x_2), f_5(x_7, x_2, x_1), f_6(x_7, x_6, x_1)\}$$

The execution of MBE(3) along lexicographical order given in example 8.1.2 is equivalent to the computation of the following expression, where we use the symbol $\downarrow$ as a short-hand for *min*

$$((f_3 + (f_1 + f_2) \downarrow x_6) \downarrow x_5 + (f_4 + (f_6 + f_5) \downarrow x_7) \downarrow x_6) \downarrow x_4 \downarrow x_3 \downarrow x_2 \downarrow x_1$$

Note that each function appears only once in the formulae.

A *computation tree* (CT first introduced in [86]) provides a graphical view of the algebraic expression. The leaves are the original functions (arguments of the formulae) and internal nodes represent the computation of intermediate functions. If the node has only one child, the only operation performed is the elimination of one variables. Otherwise, all the children are summed and one variable is eliminated. Figure A.1.*a* depicts the CT of the previous example. Dotted lines emphasize tree-leaves, which are associated to original functions. Adjacent to each internal node we indicate the variable that is eliminated. Although CTs are somehow related to decomposition-trees, they differ in the way they represent original functions. Besides, since CTs originate from MBE executions, they do not need to satisfy the running intersection property [57].

In the following we distinguish the computation of the CT from the evaluation of its associated expression. Given the scope of the original functions, a variable ordering, a policy for mini-bucket partitioning and a value for z, it is possible to compute the corresponding CT as a pre-process. Computing the CT is no more than finding the set of computations that the algorithm will perform in order to evaluate the formula.

One advantage of computing the CT in a pre-process is that it makes it easy to obtain the exact memory demands of MBE by summing the space

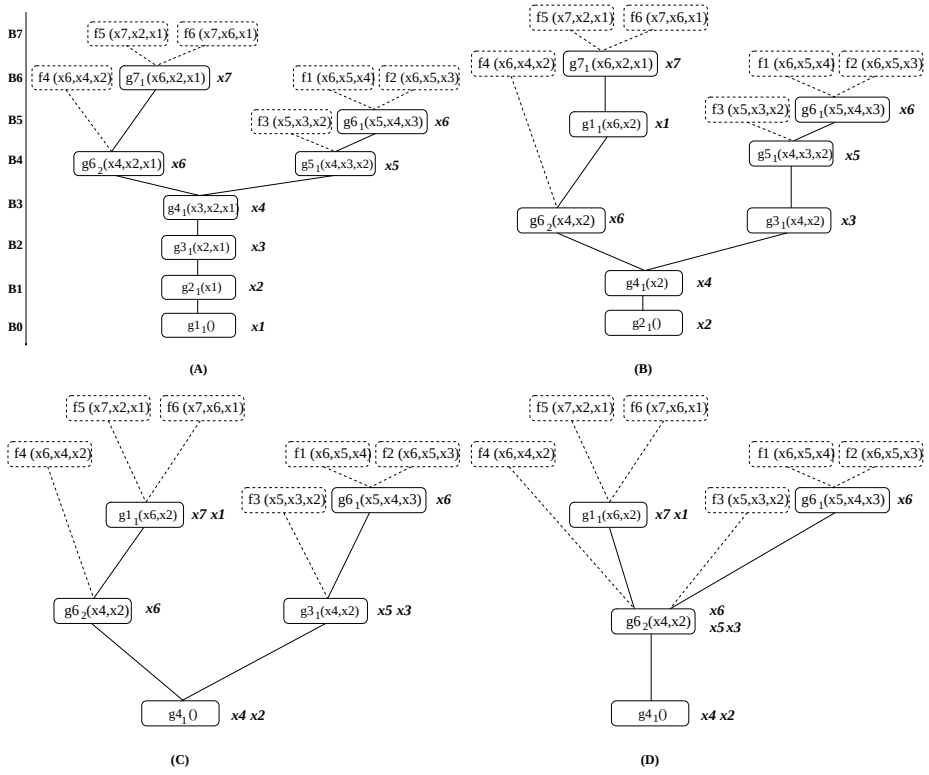

Figure A.1: Four different computation trees: A) original CT, B) after branch re-arrangement, C) after vertical compaction, D) after horizontal compaction

requirements of every internal node of the CT[1]. For instance, the CT in Figure A.1.A, will need to store 5 functions of arity 3, 1 functions of arity 2, 1 function of arity 1 and 1 function of arity 0. Assuming domains of size 10, MBE will need to store $5 \times 10^3 + 1 \times 10^2 + 1 \times 10^1 + 1 \times 10^0 = 5111$ table entries.

[1]Even when original function are given explicitely as tables, do not include their space in the MBE cost.

A.1.2 Local Transformations

CTs allow us to identify and remedy some space inefficiencies of MBE. In the following we describe two local transformations of CTs that improve their space requirements.

Branch Re-arrangement

Consider again the CT in Figure A.1.*a*. Observe that if we follow any branch in top-down order, variables are eliminated in decreasing order, because this is the order used by MBE. As a consequence, the elimination of x_1 is left to the end. However, this variable only appears in the two leftmost leaves. It is inefficient to carry it over down to the CT root, since it could have been eliminated higher up.

Consider a node v of a CT with a single child. Let x_i be the variable that is eliminated at v. Let u be the first descendent of v with $k > 1$ children. If only one child w of u has x_i in its scope, node v (namely, the elimination of x_i) can be moved in between w and u. We only perform the change if w is not a leaf. *Branch re-arrangement* is the process of applying the previous rule in a bottom-up order, moving nodes as close to the leaves as possible. The benefit of branch re-arrangement is that x_i disappears from the scope of intermediate functions earlier in the tree. In the CT of Figure A.1.*a*, the leftmost branch can be re-arranged: variable x_1 can be eliminated right after x_7. Moreover, the rightmost branch can also be re-arranged: variable x_3 can be eliminated right after x_5. Figure A.1.*b* shows the resulting CT. The space requirements of the new CT are decreased from 5111 to 3311. Observe that branch re-arrangement can never increase the space requirements of a CT.

Vertical Compaction

Consider the CT in Figure A.1.*b*. There are two single-child nodes. In single-child nodes the only associated computation is a variable elimination. MBE considers each variable elimination as an independent operation because they

take place in different buckets. However, a sequence of variable eliminations can be performed simultaneously in a single step without changing the outcome or increasing the time complexity. The advantage is that intermediate functions do not need to be stored.

Vertical compaction is the process of merging *internal linear paths* into one node representing the sequence of computations. An internal linear path is a path between an internal node v and one of its ancestors w, $(v, v_1 \ldots, v_k, w)$, such that every node in the path except v has only one child. After the compaction every internal node of the CT has $k > 1$ children. There is one exception: there may be internal nodes with only child if the child is a leaf. Figure A.1.*c* depicts the result of applying vertical compaction to the CT of Figure A.1.*b*. The space requirements of the new CT are decreased from 3311 to 1301. It is clear that the compaction of a CT may produce space saving and can never increase the space requirements of a CT.

A.1.3 Depth-First MBE

A CT can be traversed in any top-down order. A node can be computed as soon as all its children are available. Whatever traversal strategy is used it has to keep all intermediate functions because they are used in the second phase of the algorithm in order to compute the upper bound. However, the space consumption of the traversal can be drastically reduced if we sacrifice the upper bound and deallocate the memory used by intermediate functions when they become redundant. A function becomes redundant as soon as its parent has been computed. Note that an alternative solution that we do not explore in this paper is to store redundant functions in the hard-disk. Thus, the upper bound is not lost.

Without memory deallocation the traversal order has no effect on the space complexity, but this is no longer true when memory is deallocated. Traversing the CT depth-first has the advantage of only demanding the space of the current branch: computing a node only requires to have available its

children, so they have to be sequentially and recursively computed. We denote by dfMBE the algorithm that traverses depth-first the CT and deallocates memory when intermediate functions become redundant. The space complexity of dfMBE can be formalized by means of a recurrence. Let v be a node, g_v the associated function and $(w_1, \ldots, w_k)$ its ordered set of children. $R(v)$ is the space complexity of computing the sub-tree rooted by a CT node v and is given by,

$$R(v) = \max_{i=1}^{k+1}\{\sum_{j=1}^{i-1} sp(g_{w_j}) + R(w_i)\}$$

where $R(w_{k+1}) = sp(g_v)$ by definition. Also, the space $sp()$ of original functions is 0 because we do not count it as used by the algorithm. The space complexity of dfMBE is obtained by evaluating $R(v)$ at the root of the CT. In words, the recursion indicates that the space required to compute node v is the maximum among the space required to compute its children. However, when computing a given child, the space occupied by all its previous siblings must be added because they need to be available for the final computation of v.

Consider the CT of Figure A.1.*c*. We showed in the previous Section that, with no memory deallocation, the space cost of internal nodes was 1301. If the CT is traversed depth-first, the cost (disregarding original functions) is,

$$\max\{R(g_{6_2}), sp(g_{6_2}) + R(g_{3_1}), sp(g_{6_2}) + sp(g_{3_1}) + sp(g_{4_1})\} =$$

$$\max\{200, 100 + 1100, 100 + 100 + 1\} = 1200$$

Observe that the order of children affects the space complexity of dfMBE. For instance, if we reverse the two children of the root in Figure A.1.*c*, the space complexity of dfMBE is decreased to,

$$\max\{R(g_{3_1}), sp(g_{3_1}) + R(g_{6_2}), sp(g_{3_1}) + sp(g_{6_2}) + sp(g_{4_1})\} =$$

$$\max\{1100, 100 + 200, 100 + 100 + 1\} = 1100$$

In our implementation of dfMBE we make an additional optimization of the CT by processing nodes from leaves to the root. At each node, we swap the order of two of its children if it brings a space improvement.

Consider now the two children of the root-node in the CT of Figure A.1.*c*. The scope of the associated functions g_{6_1} and g_{6_2} is the same. Since they will be summed up, one table can be shared to stored both of them as follows: the table entries are initialized to 0, the two functions are computed sequentially, and each function value is added to the table current value. Figure A.1.*d* illustrates this idea. The cost of dfMBE with this new CT is,

$$\max\{R(g_{6_2}), sp(g_{6_2}) + sp(g_{4_1})\} =$$

$$\max\{\max\{100, 100+100, 100+1000, 100+1000\}, 100+1\} = 1100$$

which brings no gain over the CT in Figure A.1.*C*. However, in some cases it may bring significant benefits. Note that $R(g_{6_2}) = \max\{R(g_{1_1}), sp(g_{1_1}) + sp(g_{6_2}), sp(g_{6_2}) + R(g_{6_1}), sp(g_{6_2}) + sp(g_{6_1})\}$. In our implementation, we check siblings pair-wise. If sharing their storing table produces space savings we take such an action.

A.1.4 Experimental Results

We have tested our approach in three different domains. We compare the memory requirements for MBE, MBE' (i.e, mini-buckets under the computation tree resulting from branch re-arrangement and vertical compaction), and dfMBE in a given computer (in other words, with a fixed amount of memory). For each domain we execute MBE(z_1), MBE'(z_2) and dfMBE(z_3), where z_1, z_2 and z_3 are the highest feasible values of the control parameter for each algorithm, given the available memory.

In all our experiments, the original CT was obtained assuming a MBE execution in which the order of variable elimination was established with the *min-degree* heuristic. For the elimination of each variable, mini-buckets are constructed one by one with the following process: Select one original

function (or a non-original function if there are no original functions left). Choose among the remaining functions the one that adds the least number of variables to the mini-bucket until no more functions can be included in that mini-bucket.

In our benchmarks domain sizes range from 2 to 44, and some instances have variables with different domain size. Consequently, the arity of a function is not a proper way to indicate its spacial cost, which means that the control parameter z of MBE may be misleading (it forbids a function of arity $z+1$ with binary domains and allows a function of arity z with domains of size 4 that is much more costly to store). We overcome this problem by modifying the meaning of z: In the original formulation of MBE, the *arity* of intermediate functions is bounded by z, but in our implementation the *size* of intermediate functions is bounded by 2^z.

Scheduling of an Earth Observation Satellite

For our first experiment, we consider instances from the Scheduling of an Earth Observation Satellite benchmark (see Appendix B.2 for a detailed description). We consider the original mono-objective description of the instances, disregarding the capacity constraint imposed by the on-board storage limit on multi orbit instances. Figure A.2 reports the results that we have obtained assuming a computer with a memory limit of 1.5 Gigabytes. The first column identifies the instance. The second column indicates the induced width with the min-degree ordering. The third, fourth and fifth columns report the memory requirements in Megabytes with the three algorithms for different values of z. If the number is given in italics it means that it surpasses the space limit of the computer and the algorithm could not be executed (the memory requirement was obtained from the analysis of the CT). The sixth and seventh column indicate the value of z and the lower bound that is obtained. For each instance, we report results for three increasing values of z: the limit for MBE, MBE' and dfMBE. It can be observed that MBE' requires from 2 to 10 times less memory than MBE, which

Instance	w^*	Memory Requirement (Mb)			z	Lower Bound
		CT_{MBE}	$CT_{MBE'}$	CT_{dfMBE}		
1504	43	*17161*	*10373*	1052	27	158274
		1945	911	65	23	148259
		1240	577	34	22	142257
1506	51	*227707*	*50435*	1310	27	180356
		5503	1099	24	21	180316
		1185	214	4	18	166305
1401	156	*137825*	*11250*	524	26	210085
		11430	874	40	22	203083
		1286	131	6	19	196080
1403	156	*237480*	*28144*	1048	27	223189
		13416	1277	36	22	193185
		1153	125	5	18	189180
1405	156	*325213*	*54378*	1179	27	219302
		7226	1317	24	21	214283
		1548	289	3	18	203268
28	139	*113739*	*14764*	1048	27	141105
		8374	1424	65	23	141105
		694	109	5	19	148105
42	51	*22558*	*6032*	1572	28	125050
		2112	1123	147	24	135050
		1090	590	65	23	133050
5	83	*82823*	*38425*	917	27	206
		1861	843	16	21	192
		536	253	4	18	186
408	60	*17903*	*7966*	1048	27	5197
		2609	1355	163	24	6195
		1408	752	5	23	5197
412	61	*58396*	*24513*	1179	27	14258
		2771	882	40	22	17224
		1420	434	16	21	14220
414	144	*172071*	*24566*	1048	27	19295
		8605	1205	49	22	18301
		1154	166	4	19	18292
505	39	*15833*	*8644*	1067	27	18231
		2834	1534	139	24	19217
		1488	800	65	23	19206
507	91	*76346*	*16932*	1310	27	15286
		6222	1571	81	23	15280
		1217	250	10	20	12255
509	151	*130553*	*26671*	1114	27	18286
		6812	1008	40	22	17285
		946	162	4	19	17267

Figure A.2: Spot5 results. Memory bound of 1.5 Gb.

allows the execution with values of z up to 4 units higher. However, the most impressive results are obtained with dfMBE, which may require 275 times less space than MBE (e.g. instance 1405). As a consequence dfMBE can be executed with values of z up to 9 units higher (e.g. instance 1506), which in turn yields lower bounds up to 20% higher (e.g. instance 507). The mean space gain from MBE to dfMBE is 113.34, the mean increment of z is 7 and the mean increment of the lower bound is 8.74%.

Probabilistic Reasoning

We tested the performance of our scheme for solving the *most probable explanation* (MPE) task on two types of belief networks: Random and Noisy-OR Networks (for a detailed description of the benchmark see Appendix B.6).

Figure A.3 present results of random and noisy-OR networks assuming a computer with a memory limit of 512 Megabytes. In each table we fix parameters N, K and P and change the value of C in order to control the network's sparseness. We always assumed empty evidence and report mean values.

It can be observed that dfMBE requires from 15 to 29 times less memory than MBE, which allows the execution with values of z up to 3 units higher. The mean space gain from MBE to dfMBE is 18.56, the mean increment of z is 3.51 and the mean increment of the lower bound is 5.75%. For uniform random networks we also report the mean number of instances executed with $CT_{MBE'}$ and CT_{dfMBE} in which the lower bound increases with respect its execution with CT_{MBE} and $CT_{MBE'}$, respectively (i.e., $\%better$ column).

With random networks we also executed the efficient WCSP branch-and-bound solver Toolbar initializing its upper bound with the lower bound given by dfMBE and observed that it did not terminate with a time limit of one hour. Considering that dfMBE with the highest z value takes less than 300 seconds in this domain, we conclude that dfMBE is a better approach than iterative deepening branch and bound.

We observed that noisy-OR networks could be easily solved to optimality

Uniform Random Bayesian Networks							
		Memory Requirement (Mb)				Lower	
N, C, P	w^*	CT_{MBE}	$CT_{MBE'}$	CT_{dfMBE}	z	Bound	% better
128, 85, 4	31.71	*3635*	*598*	239	26.36	18.61	40
		2579	315	171	25.75	18.35	90
		370	84	45	22.95	17.56	-
128, 95, 4	43.96	*4144*	*999*	205	26.21	20.68	50
		1941	317	146	24.9	20.34	90
		335	94	43	22.5	19.51	-
128, 105, 4	38.71	*4537*	*825*	264	26.2	23.58	60
		2192	391	185	25.3	23.27	95
		358	89	48	22.7	22.16	-
128, 115, 4	48.32	*4114*	807	261	25.85	26.22	60
		1823	345	172	24.7	25.61	100
		355	99	43	22.5	24.69	-

Noisy-OR $P_{noise} = 0.40$						
		Memory Requirement (Mb)				
N, C, P	w^*	CT_{MBE}	$CT_{MBE'}$	CT_{dfMBE}	z	% solved
128, 85, 4	35.39	*4777*	*662*	164	26.35	73
		2805	256	153	25.6	68
		331	68	29	22.65	47
128, 95, 4	38.61	*4331*	*681*	222	26.25	84
		2545	308	169	25.35	84
		340	74	34	22.55	58
128, 105, 4	43.06	*3125*	*683*	260	25.55	50
		1646	285	136	24.6	50
		364	91	45	22.45	15
128, 115, 4	46.51	*4446*	*918*	199	25.95	65
		1530	352	149	24.75	50
		340	102	46	22.55	25
Noisy-OR $P_{noise} = 0.50$						
128, 85, 4	40.74	*4780*	*631*	242	26.45	75
		3154	330	177	25.7	75
		384	71	33	22.8	60
128, 95, 4	38.12	*3663*	356	243	25.89	55
		2170	309	158	25.15	55
		368	76	49	22.63	25
128, 105, 4	43.04	*5080*	*952*	245	26.4	65
		2006	329	109	24.8	65
		371	79	33	22.6	45
128, 115, 4	46.25	*3506*	*964*	227	26.05	60
		1552	342	176	24.7	45
		384	94	43	22.5	35

Figure A.3: MPE on bayesian networks. 20 samples. Memory bound of 512 Mb.

Instance	w^*	Memory Requirement (Mb)			z
		CT_{MBE}	$CT_{MBE'}$	CT_{dfMBE}	
graph05	135	*15955*	*1992*	49	28
		12880	1102	86	27
		1483	201	25	24
graph06	296	*30364*	*2544*	300	28
		17291	1320	300	27
		1354	117	10	23
graph07	146	*14797*	*1866*	527	28
		8187	266	49	27
		1511	180	45	24
graph11reduc	275	*30331*	*2044*	113	28
		15630	1183	113	27
		1267	154	22	23
graph11	495	*55260*	*3079*	22	28
		5935	338	30	25
		547	83	11	21
graph12	234	*23532*	*3570*	692	28
		3399	493	134	26
		1379	230	21	24
graph13reduc	619	*67123*	*6447*	723	28
		9964	1070	121	25
		1572	141	13	22
graph13	706	*89091*	*6828*	1067	28
		7354	515	24	25
		806	161	11	21

Figure A.4: RLFAP. Memory bound of 1.5 Gb.

with ToolBar. Therefore, we also report for each parameter setting and each value of z, how many instances are solved to optimality with MBE, MBE' and dfMBE.

Resource allocation

For our third experiment, we consider the *radio link frequency assignment problem* (see Appendix B.7 for more details). Figure A.4 reports graph instances where we obtained the best results. It can be observed that dfMBE is also very effective in this domain. It can require on average by 430.25 times less memory than MBE, which allows the execution with values of z up to 5.25 units larger.

A.2 Improving Lower Bound

In this Section we show how to increase the lower bound obtained by MBE while maintaining its space demands. The new method introduces a new propagation phase that MBE must execute at each bucket. It consists on performing an arrangement of costs before processing each bucket as follows. Mini-buckets are structured into a tree and costs are moved along branches from the leaves to the root. As a result, the root mini-bucket accumulates costs that will be processed together, while classical MBE would have processed them independently. Note that the new propagation phase does not increase the complexity with respect classical MBE.

Our experiments on *scheduling*, *combinatorial auctions* and *maxclique* show that the addition of this propagation phase increases the quality of the lower bound provided by MBE quite significatively. Although the increase depends on the benchmark, the typical percentage is 50%. However, for some instances, the propagation phase gives a dramatic percentage increment up to 1566%.

A.2.1 Preliminaries

Notation

Let f be a function and let $Y \subseteq \mathcal{X}$ be a subset of variables. Along this Section, we will use $f[Y]$ as a short-hand for $\min_{var(f)-Y}\{f\}$.

Fair c-semirings

Given a c-semiring $\mathcal{K} = (A, \oplus, \otimes)$, $\mathcal{K}$ is *fair* [28] if for any pair of valuations $a, b \in A$, with $a \leq b$, there exists a maximum difference of b and a. This unique maximum difference of b and a is denoted by $b \ominus a$. This property ensures the equivalence of a SCSP problem when the two operations $\otimes$ and $\ominus$ are applied.

Abusing notation, we extend the $\ominus$ operator from the set of valuations

$\ominus : A \times A \rightarrow A$ to the set of functions $\ominus : \mathcal{F} \times \mathcal{F} \rightarrow \mathcal{F}$. The substraction of two functions f and g, noted $f \ominus g$, is a new function that *substracts* the information of g from f.

Definition A.2.1 *Let f and g be two functions such that $var(g) \subseteq var(f)$ and $\forall t \in var(f),\ f(t) \geq g(t)$. Their* subtraction*, noted $f \ominus g$, is a new function with scope $var(f)$ defined as,*

$$(f \ominus g)(t) = f(t) \ominus g(t)$$

for all tuple $t \in var(f)$.

In [28] it is shown that the most important reasoning tasks are fair. Although our approach can be used in any fair reasoning task, for the sake of simplicity, we will focus on WCSPs.

As has been shown, in weighted CSPs (WCSPs), A is the set $\mathbb{N}^\infty$ and its $\otimes$ is the usual sum over naturals. It is easy to see that the $\ominus$ operator over $\mathbb{N}^\infty$ is the usual substraction.

Equivalence Preserving Transformations

We say that two WCSPs are equivalent if they have the same optimum. There are several transformations that preserve the equivalence. For instance, if we take any pair of cost functions $f, g \in \mathcal{F}$ from a WCSP $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ and replace them by their sum $f + g$, the result is an equivalent problem. The replacement of $\mathcal{B}$ by g performed by BE (Figure 7.1) is another example of equivalence-preserving transformation. Very recently, a new kind of WCSP transformation has been used in the context of soft local consistency [89, 27]. The general idea is to *move* costs from one cost function to another. More precisely, costs are subtracted from one cost function and added to another. Formally, let f and h be two arbitrary functions. The *movement of costs* from f to g is done sequentially in three steps:

$h := f[var(f) \cap var(g)]$

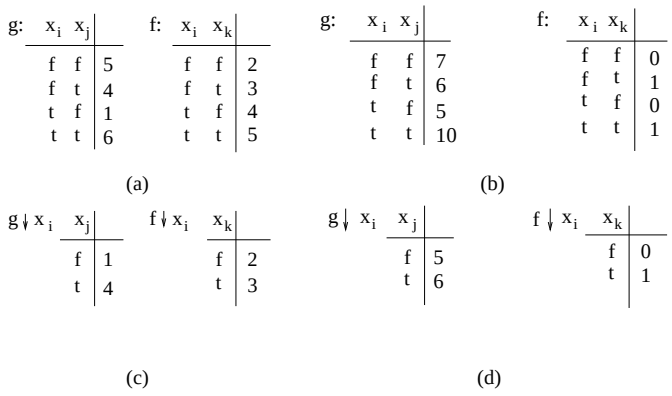

Figure A.5: Example of functions.

$f := f - h$
$g := g + h$

In words, function h contains costs in f that can be captured in terms of the common variables with g. Hence, they can be kept either in h or in f. Then, this costs are moved from f to g. The time complexity of this operation is $O(d^{\max\{|var(f)|,|var(g)|\}})$. The space complexity is the size of h stored as a table, $O(d^{|var(h)|\}})$, which is negligible in comparison with the larger function f.

Example A.2.1 *Consider the functions on Figure A.5 (a). They are defined over boolean domains and given as a table of costs. Let function h represents the costs that can be moved from function f to function g. Observe that, as f and g only share variable x_i, then $h = f[x_i]$, where $h(false) = 2$ and $h(true) = 4$. Figure A.5 (b), shows the result of moving the costs from f to g. Observe that costs of tuples t such that $var(t) = \{x_i, x_j, x_k\}$ are preserved.*

A.2.2 Bucket Propagation

The new refinement of MBE consists on performing a movement of costs in each bucket before processing it. We incorporate the concept of equivalence-preserving transformation into MBE, but only at the bucket level. The idea is to *move* costs between minibuckets aiming at a propagation effect. We pursue the accumulation of as much information as possible in one of the mini-buckets.

The following example illustrates and motivates the idea. Suppose that MBE is processing a bucket containing two functions f and g, each one forming a mini-bucket. Variable x_i is the one to be eliminated. Standard MBE would process independently each minibucket, eliminating variable x_i in each function. It is precisely this independent elimination of x_i from each mini-bucket where the lower bound of MBE may lose accuracy. Ideally (i.e, in BE), f and g should be added and their information should *travel* together along the different buckets. However, in MBE their information is split into two pieces for complexity reasons. What we propose is to transfer costs from f to g (or conversely) before processing the mini-buckets. The purpose is to put as much information as possible in the same mini-bucket, so that all this information is jointly processed as BE would do. Consequently, the pernicious effect of splitting the bucket into mini-buckets will presumably be minimized. Figure A.5 depicts a numerical illustration. Consider functions f and g from Figure A.5 (a). If variable x_i is eliminated independently, we obtain the functions in Figure A.5 (c). If the problem contains no more functions, the final lower bound will be 3. Consider now the functions in Figure A.5 (b) where costs have been moved from f to g. If variable x_i is eliminated independently, we obtain the functions in Figure A.5 (d), with which the lower bound is 5.

The previous example was limited to two mini-buckets containing one function each. Nevertheless, the idea can be easily generalized to arbitrary mini-bucket arrangements. At each bucket $\mathcal{B}$, we construct a *propagation tree* $T = (V, E)$ where nodes are associated with mini-buckets and edges represent

movement of costs along branches from the leaves to the root. Each node waits until receiving costs from all its children. Then, it sends costs to its parent. This flow of costs accumulates and propagates costs towards the root.

A.2.3 Mini-Bucket Elimination with Bucket Propagation

The refinement of MBE that incorporates the idea of bucket propagation is called MBE^p. In Figure A.6 we describe a preliminary version. A more efficient version regarding space will be discussed later on. MBE^p and MBE are very similar and, in the following, we discuss the main differences. After partitioning the bucket into mini-buckets (line 3), MBE^p computes the sum of all the functions in each mini-bucket (line 4). Next, it constructs a propagation tree $T = (V, E)$ with one node j associated to each function g_j. Then, costs are propagated (lines 6, 11-16). Finally, variable x_i is eliminated from each mini-bucket (line 7) and resulting functions are added to the problem in replacement of the bucket (line 8).

Procedure `Propagation` is also depicted in Figure A.6. Let j be an arbitrary node of the propagation tree such that has received costs from all its children. It must send costs to its parent $parent(j)$. First, it computes in function h_j the costs that can be sent from j to its parent (line 13). Then, function h_j is subtracted from g_j and summed to $g_{parent(j)}$ (lines 14 and 15). The propagation phase terminates when the root receives costs from all its children.

Observe that the previous implementation of MBE^p (Figure A.6) computes in two steps (lines 4 and 7), what plain MBE computes in one step. Consequently, MBE^p stores functions with arity up to $z+1$ while MBE only stores functions with arity up to z. Therefore, the previous description of MBE^p has a space complexity slightly higher than MBE, given the same value of z. In the following, we show how the complexity of MBE^p can

function $MBE^p(P, z)$
1. **for each** $i = n..1$ **do**
2. $\mathcal{B} := \{f \in F |\ x_i \in var(f)\}$;
3. $\{\mathcal{P}_1, \ldots, \mathcal{P}_k\} :=$ Partition$(\mathcal{B}, z)$;
4. **for each** $j = 1 \ldots k$ **do** $g_j := \sum_{f \in \mathcal{P}_j} f$;
5. $(V, E) :=$ `PropTree`$(\{g_1, \ldots, g_k\})$;
6. `Propagation`$((V, E))$;
7. **for each** $j = 1..k$ **do** $g_j := \min_{x_i}\{g_j\}$;
8. $F := (F \cup \{g_1, \ldots, g_k\}) - \mathcal{B}$;
9. **endfor**
10. **return**(g_1);

endfunction

procedure `Propagation`$((V, E))$
11. **repeat**
12. select a node j s.t it has received the messages from all its children;
13. $h_j := g_j[var(g_j) \cap var(g_{parent(j)})]$;
14. $g_j := g_j - h_j$;
15. $g_{parent(j)} := g_{parent(j)} + h_j$;
16. **until** root has received all messages from its children;

endprocedure

Figure A.6: Mini-Bucket Elimination with Propagation (preliminary version). Given a WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, the algorithm returns a zero-arity function g_1 with a lower bound of the optimum cost.

be made similar to the complexity of MBE. First, we extend the concept of movement of costs to deal with sets of functions. Let F and G be two sets of costs functions. Let $var(F) = \cup_{f \in F} var(f)$, $var(G) = \cup_{g \in G} var(g)$ and $Y = var(F) \cap var(G)$. The *movement of costs* from F to G is done sequentially in three steps:

$h := (\sum_{f \in F} f)[Y]$
$F := F \cup \{-h\}$
$G := G \cup \{h\}$

where $-h$ means that costs contained in h are to be subtracted instead of summed, when evaluating costs of tuples on F. Observe that the first step can be efficiently implemented as,

$$\underset{t \in Y}{\forall}, h(t) := \min_{(t' \in var(F) \text{ s.t. } t' = t \cdot t'')} \{\sum_{f \in F} f(t')\}$$

This implementation avoids computing the sum of all the functions in F. The time complexity of the operation is $O(d^{|var(F)|})$. The space complexity is $O(d^{|Y|})$.

Figure A.7 depicts the new version of MBE^p. The difference with the previous version is that functions in mini-buckets do not need to be summed before the propagation phase (line 4 is omitted). Procedure `Propagation` moves costs between mini-buckets preserving the set of original functions. Line 7, sums the functions in the mini-buckets and eliminates variable x_i in one step, as plain MBE would do.

Observe that the time complexity of line 13 is $O(d^{z+1})$, because $|var(\mathcal{P}_j)| \leq z + 1$ (by definition of mini-bucket). The space complexity is $O(d^z)$ because $|var(h)| \leq z$ (note that $var(\mathcal{P}_j) \neq var(\mathcal{P}_{parent(j)})$ because otherwise they would have been merged into one mini-bucket). The previous observation leads to the following result.

Theorem A.2.1 *The time and space complexity of MBE^p is $O(d^{z+1})$ and*

function $MBE^p(P, z)$
1. **for each** $i = n..1$ **do**
2. $\mathcal{B} := \{f \in F |\ x_i \in var(f)\}$;
3. $\{\mathcal{P}_1, \ldots, \mathcal{P}_k\} :=$ Partition$(\mathcal{B}, z)$;
5. $(V, E) :=$ PropTree$(\{\mathcal{P}_1, \ldots, \mathcal{P}_k\})$;
6. Propagation$((V, E))$;
7. **for each** $j = 1..k$ **do** $g_j := \min_{x_i}\{(\sum_{f \in \mathcal{P}_j} f) - h_j\}$;
8. $F := (F \cup \{g_1, \ldots, g_k\}) - \mathcal{B}$;
9. **endfor**
10. **return**(g_1);

endfunction

procedure `Propagation`$((V, E))$
11. **repeat**
12. select a node j s.t it has received the messages from all its children;
13. $h_j := (\sum_{f \in \mathcal{P}_j} f)[var(\mathcal{P}_j) \cap var(\mathcal{P}_{parent(j)})]$;
14. $\mathcal{P}_j := \mathcal{P}_j \cup \{-h_j\}$;
15. $\mathcal{P}_{parent(j)} := \mathcal{P}_{parent(j)} \cup \{h_j\}$;
16. **until** root has received all messages from its children;

endprocedure

Figure A.7: Mini-Bucket Elimination with Propagation. Given a WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, the algorithm returns a zero-arity function g_1 with a lower bound of the optimum cost.

$O(d^z)$, respectively, where d is the largest domain size and z is the value of the control parameter.

A.2.4 Computation of the Propagation Tree

In our preliminary experiments we observed that the success of the propagation phase of MBE^p greatly depends on the flow of information, which is captured in the propagation tree. In the following we discuss two ideas that heuristically lead to good propagation trees. Then, we will propose a simple method to construct good propagation trees.

For the first observation, consider MBE with $z = 1$ in a problem with four binary functions $f_1(x_1, x_2), f_2(x_2, x_3), f_3(x_2, x_4), f_4(x_3, x_4)$. Variable x_4 is the first to be eliminated. Its bucket contains f_3 and f_4. Each function forms a mini-bucket. MBE^p must decide whether to move costs from f_3 to f_4 or conversely. Observe that after the elimination of x_4, f_4 will go to the bucket of x_3 where it will be summed with f_2. Then, they will go to the bucket of x_2. However, f_3 will *jump* directly to the bucket of x_2. For this reason, it seems more appropriate to move costs from f_3 to f_4. In f_4 the costs go to a higher mini-bucket, so they have more chances to propagate useful information. One way to formalize this observation is the following: We associate to each mini-bucket $\mathcal{P}_j$ a binary number $N_j = b_n b_{n-1} \dots b_1$ where $b_i = 1$ iff $x_i \in \mathcal{P}_j$. We say that mini-bucket $\mathcal{P}_j$ is smaller than $\mathcal{P}_k$ (noted $\mathcal{P}_j < \mathcal{P}_k$) if $N_j < N_k$. In our propagation trees parents will always be larger than their children.

For the second observation, consider three functions

$$f(x_7, x_6, x_5, x_4),\ g(x_7, x_3, x_2, x_1),\ h(x_7, x_6, x_5, x_1)$$

Observe that f shares 1 variable with g and 3 with h. The number of common variables determines the arity of the function that is used as a *bridge* in the cost transfer. The narrower the bridge, the less information that can be captured. Therefore, it seems better to move costs between f and h than between f and g.

In accordance with the two previous observations, we construct the propagation tree as follows: the parent of mini-bucket $\mathcal{P}_u$ will be mini-bucket $\mathcal{P}_w$ such that $\mathcal{P}_u < \mathcal{P}_w$ and they share a maximum number of variables. This strategy combines the two criteria discussed above.

A.2.5 Experimental Results

We have tested our approach in three different domains. The purpose of the experiments is to evaluate the effectiveness of the propagation phase and the impact of the propagation tree on that propagation. To that end, we compare the lower bound obtained with three algorithms: standard MBE, MBE with bucket propagation using as a propagation tree a chain of mini-buckets randomly ordered (i.e., MBE_r^p), and MBE with bucket propagation using a propagation tree heuristically built as explained in Section A.2.4 (i.e., MBE_h^p). For each domain, we execute those three algorithms with different values of the control parameter z in order to analyze its effect (the highest value of z reported is the highest feasible value given the available memory). In all our experiments, the order of variable elimination is established with the *min-fill* heuristic. All the experiments are executed in a Pentium IV running Linux with 2Gb of memory and 3 GHz.

Scheduling of an Earth Observation Satellite

For our first experiment, we consider the scheduling of an earth observation satellite (see Appendix B.2 for a detailed description). We experiment with instances modelled as mono-objective optimization problems. Moreover, we discard the capacity constraint on multi orbit instances.

Figure A.8 shows the results. The first column identifies the instance. The second column indicates the value of the control parameter z with which the algorithms are executed. Columns third and fourth report the lower bound obtained and the execution time for standard MBE, respectively. Columns fifth and sixth indicates for MBE_r^p the percentage increment of the lower

Instance	z	MBE(z)		MBE^p_r(z)		MBE^p_h(z)	
		Lb.	Time(sec.)	%	Time(sec.)	%	Time(sec.)
1506	20	184247	827.63	1.6	1628.93	29.8	1706.6
	15	163301	25.43	-5.5	51.48	30.6	51.39
	10	153274	1.33	-13.7	2.65	21.5	2.64
1401	20	184084	691.08	16.8	1469.36	58.6	1574.26
	15	170082	20.82	4.7	47.35	45.8	46.92
	10	155075	1.02	-10.3	2.13	53.5	2.17
1403	20	181184	814.55	7.1	1702.82	59.6	1919.48
	15	162170	27.82	7.3	55.94	57.3	56.9
	10	146155	1.3	10.9	2.58	60.2	2.6
1405	20	191258	1197.06	0.5	2537.64	42.3	2622.88
	15	169233	33.88	-2.3	93.88	54.9	81.17
	10	142206	1.7	-25.3	3.51	64.7	3.5
1407	20	191342	1415.91	-4.0	2935.78	53.8	3008.78
	15	166298	47.44	3.5	94.17	60.1	102.78
	10	144264	2.03	13.8	4.19	68.6	4.23
28	20	134105	252.14	2.2	500.97	38.0	510.72
	15	121105	7.77	-1.6	15	52.8	16.16
	10	103105	0.36	16.4	0.71	49.4	0.71
29	20	8058	4.92	-0.01	5.3	0.01	5.32
	15	8055	0.28	-0.1	0.34	0.02	0.34
	10	8050	0.01	-0.01	0.02	0.07	0.02
408	20	5212	51.19	19.1	75.39	19.3	72.5
	15	5200	2.11	18.7	3.29	19.3	3.41
	10	2166	0.11	38.1	0.2	139.0	0.2
412	20	17314	167.91	5.4	278.29	40.5	278.7
	15	15270	6.49	6.2	10.98	72.1	11.1
	10	10233	0.27	87.8	0.5	88.4	0.78
414	20	23292	629.36	-12.9	1278.39	17.4	1306.98
	15	18268	20.14	-16.3	42.87	49.4	42.99
	10	16213	1.05	-31.0	2.35	49.8	2.09
42	20	127050	38.9	-4.7	71.47	7.8	68.35
	15	111050	1.43	-1.8	2.52	14.4	2.55
	10	93050	0.06	2.1	0.12	19.3	0.12
505	20	19240	51.36	-36.3	66.9	5.2	63.16
	15	16208	2.2	-18.5	3.35	0.1	3.23
	10	13194	0.15	-15.2	0.21	15.1	0.21
507	20	16292	276.74	-6.1	510.66	0.2	520.3
	15	14270	9.84	6.7	19.01	42.2	18.88
	10	11226	0.47	8.6	0.92	53.7	0.92
509	20	22281	507.64	4.6	1026.43	22.5	1046.89
	15	20267	16.2	-24.6	34.68	34.7	34.72
	10	14219	0.83	14.0	1.64	77.7	1.62

Figure A.8: Experimental results on Spot5 instances.

bound measured as $((Lb_{MBE_r^p} - Lb_{MBE})/Lb_{MBE}) * 100$ and the execution time. Columns seventh and eighth reports the same information for MBE_h^p.

The first thing to be observed is that the results obtained with MBE_r^p does not follow a clear tendency. MBE_r^p increases and decreases the lower bound obtained with standard MBE almost the same times. However, MBE_h^p increases the lower bound obtained with MBE for all the instances. Moreover, when both MBE_r^p and MBE_h^p increase the lower bound, MBE_h^p is always clearly superior. Therefore, it is clear that an adequate propagation tree impacts on the bounds obtained.

Regarding MBE_h^p, it increases up to 139% the lower bound with respect MBE (e.g. instance 408). The mean increment is 54%, 38%, and 28% when the value of the control parameter z is 10, 15, and 20, respectively. Note that the effect of the propagation is higher for lower values of z because, as we increase the value of z, the number of functions in each mini-bucket increases and the number of mini-buckets decreases. Therefore, the propagated information also decreases and the effect of the propagation is diminished. Moreover, the lower bounds obtained with MBE_h^p and z set to 10 outperforms the ones obtained with MBE and z set to 20 in almost all the instances, which means that the time and space required for obtaining a bound of a given quality is decreased.

Regarding cpu time, MBE_h^p is from 2 to 3 times slower than MBE. The reason is that cost functions are evaluated twice: the first one during the propagation phase for establishing the costs to be moved, and the second one during the regular process of variable elimination. However, it is important to note that it is the space and not the time what bounds the maximum value of z that can be used in practice. As a consequence, that constant increase in time is not that significant as the space complexity remains the same.

Combinatorial Auctions

We experiment on instances from the combinatorial auctions benchmark for the *path* and *regions* model (for a detailed description see Appendix B.1).

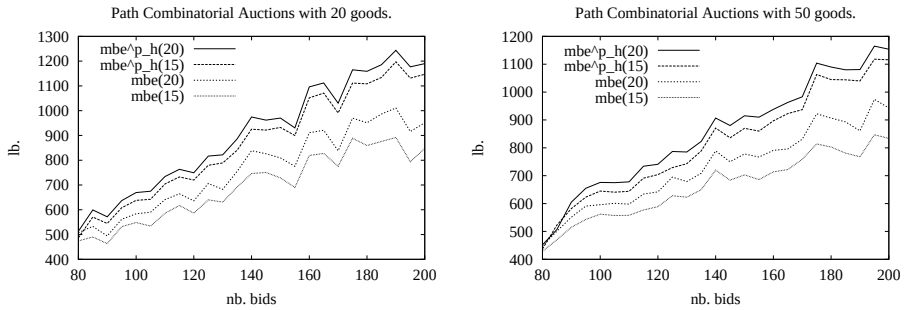

Figure A.9: Combinatorial Auctions. Path distribution.

We execute algorithms MBE, MBE_r^p, and MBE_h^p with z equal to 15 and 20. We do not report results with MBE_r^p because it was always very inferior than MBE_h^p. Moreover, we only report results on the *path* model because the results for the *regions* model follows the same pattern.

Figure A.9 reports the results for *path* instances with 20 and 50 goods, respectively. As can be observed, the behaviour for both configurations is almost the same. Regarding the algorithms, it is clear that MBE_h^p always outperformes MBE. Note that the lower bound obtained with $MBE_h^p(z = 15)$ is clearly superior than that obtained with $MBE(z = 20)$. Moreover, as pointed out in the previous domain, the effect of the propagation in each sample point is higher for $z = 15$ than for $z = 20$. That is, the percentage of increment in the lower bound obtained with $MBE_h^p(z = 15)$ is higher than that of $MBE_h^p(z = 20)$. Finally, it is important to note that the impact of the propagation is higher when the problems become harder (i.e., as the number of bids increase).

Maxclique

We test our approach on the maxclique benchmark (see Appendix B.5 for more details). Figures A.10 and A.11 report the results. The first column identifies the instance. The second column indicates the value of the control parameter z with which the algorithms are executed. The third column re-

Instance	z	MBE	MBE_r^p	MBE_h^p
		Lb.	%	%
brock200-1	18	66	30.3	48.4
	10	51	52.9	78.4
brock200-2	18	55	67.2	103.6
	10	29	200	268.9
brock200-3	18	64	48.4	68.7
	10	38	139.4	173.6
brock200-4	18	63	36.5	65.0
	10	41	121.9	131.7
brock400-1	18	79	100	141.7
	10	46	256.5	273.9
brock400-2	18	75	114.6	157.3
	10	44	261.3	277.2
brock400-3	18	87	88.5	114.9
	10	44	250	286.3
brock400-4	18	76	106.5	160.5
	10	47	248.9	289.3
brock800-1	18	71	336.6	454.9
	10	41	675.6	773.1
brock800-2	18	63	395.2	520.6
	10	37	748.6	875.6
brock800-3	18	68	352.9	483.8
	10	44	604.5	706.8
brock800-4	18	71	343.6	460.5
	10	36	758.3	902.7
c-fat200-1	18	71	32.3	78.8
	10	62	27.4	112.9
c-fat200-2	18	63	38.0	82.5
	10	48	77.0	156.2
c-fat200-5	18	55	23.6	12.7
	10	37	32.4	70.2
c-fat500-10	18	77	115.5	123.3
	10	52	173.0	253.8
c-fat500-1	18	132	84.0	137.1
	10	107	126.1	196.2
c-fat500-2	18	108	108.3	164.8
	10	85	160	254.1
c-fat500-5	18	83	145.7	202.4
	10	74	163.5	264.8
hamming10-2	18	412	-66.9	-72.0
	10	419	-72.0	-73.7
hamming10-4	18	119	264.7	413.4
	10	77	451.9	720.7
hamming6-2	18	32	-28.1	-31.2
	10	32	-50	-59.3
hamming6-4	18	45	-4.4	2.2
	10	33	9.0	33.3
hamming8-2	18	114	-59.6	-64.9
	10	113	-74.3	-78.7
hamming8-4	18	82	46.3	89.0
	10	51	113.7	215.6

Instance	z	MBE	MBE_r^p	MBE_h^p
		Lb.	%	%
johnson16-2-4	18	72	-4.1	11.1
	10	56	10.7	48.2
johnson32-2-4	18	195	27.6	71.2
	10	134	75.3	150
johnson8-2-4	18	23	-4.3	0
	10	20	-20	-5
johnson8-4-4	18	45	-22.2	-11.1
	10	40	-15	-10
keller4	18	70	27.1	54.2
	10	41	97.5	168.2
keller5	18	90	246.6	394.4
	10	61	414.7	634.4
MANN-a27	15	247	0.4	0.4
	10	244	-1.2	0.8
MANN-a45	15	677	-0.7	0.4
	10	671	-0.1	0.1
MANN-a81	15	2177	0.0	0.3
	10	2171	-0.1	0.5
p-hat1000-1	15	85	380	654.1
	10	63	577.7	873.0
p-hat1000-2	15	57	589.4	821.0
	10	36	1013.8	1325
p-hat1000-3	15	82	364.6	415.8
	10	50	668	764
p-hat1500-1	15	69	802.8	1292.7
	10	82	686.5	1021.9
p-hat1500-2	15	64	812.5	1112.5
	10	45	1226.6	1566.6
p-hat1500-3	15	79	624.0	706.3
	10	54	924.0	1111.1
p-hat300-1	18	62	112.9	195.1
	10	48	187.5	306.2
p-hat300-2	18	61	121.3	168.8
	10	38	247.3	328.9
p-hat300-3	18	76	71.0	100
	10	51	145.0	172.5
p-hat500-1	18	74	170.2	301.3
	10	50	330	524
p-hat500-2	18	75	178.6	248
	10	39	407.6	556.4
p-hat500-3	18	93	125.8	169.8
	10	50	300	338
p-hat700-1	15	66	340.9	581.8
	10	52	482.6	711.5
p-hat700-2	18	63	357.1	492.0
	10	36	672.2	919.4
p-hat700-3	18	78	260.2	330.7
	10	44	543.1	588.6
san1000	15	89	319.1	493.2
	10	100	260	438

Figure A.10: Experimental results on maxclique instances.

Instance	z	MBE	MBE_r^p	MBE_h^p
		Lb.	%	%
san200-0.7-1	18	69	26.0	53.6
	10	50	82	86
san200-0.7-2	18	84	40.4	51.1
	10	53	75.4	115.0
san200-0.9-1	18	108	-1.8	0
	10	82	18.2	14.6
san200-0.9-2	18	85	20	17.6
	10	68	25	27.9
san200-0.9-3	18	83	21.6	18.0
	10	67	34.3	26.8
san400-0.5-1	18	79	115.1	194.9
	10	58	189.6	289.6
san400-0.7-1	18	84	95.2	144.0
	10	55	138.1	209.0

Instance	z	MBE	MBE_r^p	MBE_h^p
		Lb.	%	%
san400-0.7-2	18	78	105.1	158.9
	10	42	247.6	309.5
san400-0.7-3	18	73	138.3	180.8
	10	47	225.5	287.2
san400-0.9-1	18	97	63.9	75.2
	10	75	93.3	98.6
sanr200-0.7	18	61	42.6	63.9
	10	45	80	104.4
sanr200-0.9	18	77	12.9	23.3
	10	61	31.1	37.7
sanr400-0.5	18	67	152.2	223.8
	10	32	406.2	543.7
sanr400-0.7	18	76	103.9	152.6
	10	47	231.9	270.2

Figure A.11: Experimental results on maxclique instances.

port the lower bound obtained with standard MBE. Columns fourth and fifth indicates, for MBE_r^p and MBE_l^p, the percentage of increment in the lower bound with respect MBE, respectively. As the behaviour of the cpu time is the same as for the previous benchmark, we do not report this information.

MBE_r^p increases the lower bound obtained with standard MBE for all the instances except for those of *hamming* and *johnson*. The percentage of increment is up to 1226% when the value of the control parameter z is 10, and up to 812% when z is the highest value. The best results are obtained with MBE_h^p which obtains a percentage increment of 1566% (see instance *p-hat1500-2*). In this case, the increase ranges from 14.6% to 1566% when z is set to 10, and from 17.6% to 1292% for the highest value of z.

It is important to note that the bound obtained with MBE_h^p is always higher than that of MBE_r^p. For some instances, the percentage of increment of MBE_h^p is more than 4 times higher the one obtained with MBE_r^p (e.g. instance *c-fat200-1*). Therefore, it is clear that an adequate propagation tree impacts on the propagation phase and, as a consequence, on the bounds obtained.

Appendix B

Benchmarks

Briefly, the following table shows the benchmarks used throughout this Thesis and the type of problems each one is concerned with.

Name	Multi-objective optimization	Mono-objective optimization	Constraint satisfaction
Combinatorial Auction	√	√	√
Scheduling of an EOS[1]	√	√	√
Max-SAT-ONE	√		
Minimum Vertex Cover	√		
Max-Clique		√	
MPE in Belief Network		√	
RLFA[2]		√	

In the following we describe in more detail each benchmark. For each one, we describe the problem(s) solved, the included instances along with their important structural properties (i.e., induced width and bandwidth) and the encoding of the corresponding reasoning task(s).

[1]Earth Observation Satellite
[2]Radio Link Frequency Assignment

The induced width of each instance is obtained under the variable ordering given by the min-degree heuristic [35]. We note its value by w^*. The bandwidth of each instance is obtained under three different variable ordering heuristics: lexicographical (noted b*(lex)), greedy min-fill [35] (noted b*(min-fill)), minimal width order [50] (noted b*(MWO)), and minimal triangulation [125] (noted b*(LEX-M)).

B.1 Combinatorial Auctions

Description

Combinatorial auctions (CA) [127] allow bidders to bid for indivisible subsets of goods. Given a set of n goods $\{1, 2, \ldots, n\}$ presented in an auction, the bidders generate m bids. Bid i is defined by the subset of requested goods $B_i \subseteq \{1, 2, \ldots, n\}$ and the money offer m_i. The *combinatorial auction problem* consists in accepting a subset of bids such that the benefits of the bid taker are maximized. Note that the same good can appear in different bids. Hence, only a bid containing each good can be accepted. This problem is a NP-Hard problem.

The combinatorial auction problem can be extended to consider more than one objective. Risk-conscious auctions (RCA) [63] are combinatorial auctions in which the bid-taker wants to control the risk of bid withdrawal following winner determination, because it may cause large losses in revenue. In this context, bid i is defined by a subset of goods $B_i \subseteq \{1, 2, \ldots, n\}$, the money offer m_i and the probability of succesful payment s_i. The *risk-conscious auction problem* consists in determining the bids accepted such that they maximize both the benefits and the succesful of payment.

Finally, in some experiments, we consider a constraint satisfaction (or decision) version of the risk-conscious auction problem in which the bid-taker provides two constants (P, R) indicating that she wants to obtain a benefit larger than P and a probability of full payment higher than R.

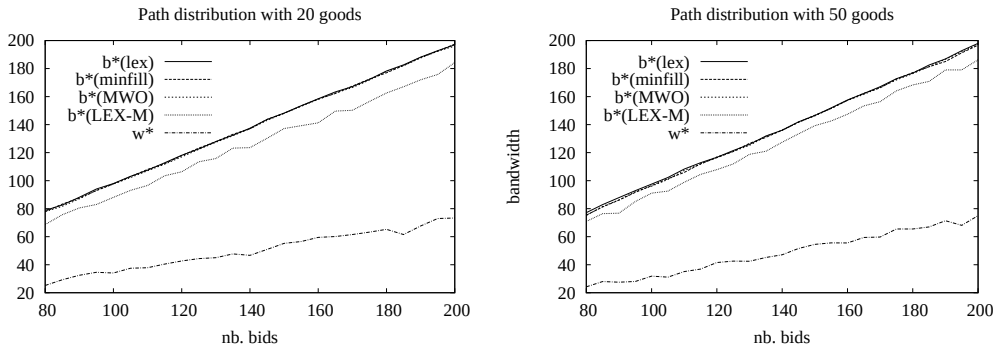

Figure B.1: Mean values of the the induced width and bandwidth for combinatorial auctions instances.

Instances

We generated (mono-objective) CA instances using the CATS generator [78]. This generator is able to generate problems with different distributions that represent real life type of problems. In our experiments, we use the *path* model. We generated instances with 20 and 50 goods, varying the number of bids from 80 to 200. For each parameter configuration, we generate samples of size 25. Figure B.1 shows the mean values of the induced width and bandwidth using the lexicographical, min-fill, MWO and LEX-M variable orderings.

Finally, we generated decision instances from the previous risk-conscious auctions. For each instance, we established the values of (P, R) in such a way that i) the instance admits a solution and ii) a small decrease of either one renders the problem unsoluble. Consequently, the instances are difficult with respect to the two constraints.

Encoding

Mono-objective CA instances can be encoded as a minimization WCSP problems, as follows. Bid i is represented by a variable x_i. Then, the set of

variables $\mathcal{X}$ is $\{x_1, \ldots, x_m\}$ taking domain values over a common domain $D_i = \{1, 0\}$ (meaning *accepted* and *discarded*, respectively). Bid incompatibilities can be modeled as $0/\infty$-functions between variables. If bid i shares a good with bid j (i.e., $\mathcal{B}_i \cap \mathcal{B}_j \neq \emptyset$), then there exists a binary function,

$$c_{ij}(x_i, x_j) = \begin{cases} 0 & \overline{x_i \wedge x_j} \\ \infty & otherwise \end{cases}$$

Finally, we can rephrase the objective of maximizing the bid taker benefit as to minimize the revenue of discarded bids. Then, the money from bid i can be modelled as a unary function,

$$b_i(x_i) = m_i(1 - x_i)$$

The objective function is $F_1(\mathcal{X}) = \sum c_{ij} + \sum b_i$.

The second objective of RCA instances can be encoded as a minimization WCSP problem, as follows. First, the probability of successful payment s_i under an assumption of probabilistic independence can be modeled as a unary function,

$$p_i(x_i) = \begin{cases} s_i & x_i = 1 \\ 1 & otherwise \end{cases}$$

Note that the objective function to be maximized is $\prod_{i=1}^{m} p_i$. After a logarithmic transformation, the objective function is additive and it has to be minimized. Formally, unary functions are,

$$p'_i(x_i) = \begin{cases} -\log s_i & x_i = 1 \\ 0 & otherwise \end{cases}$$

and the objective function is $F_2(\mathcal{X}) = \sum_{i=1}^{m} p'_i$. Finally, since both objective functions of RCA instances can be expressed as WCSP problems, they can be modeled as MO-WCSP problems as described in Chapter 4. Briefly, the set of variables $\mathcal{X}$ is $\{x_1, \ldots, x_m\}$, the set of domain values is $D_i = \{1, 0\}$,

and the set of multi-objective cost functions is,

$$c_{ij}(x_i, x_j) = \begin{cases} \{(0,0)\} & \overline{x_i \wedge x_j} \\ \{(\infty,\infty)\} & otherwise \end{cases}$$

$$b_i(x_i) = \{(m_i(1-x_i), 0)\}$$

$$p'_i(x_i) = \begin{cases} \{(0, -\log s_i)\} & x_i = 1 \\ \{(0,0)\} & otherwise \end{cases}$$

Finally, the decision version of RCA problems can be modeled as a CSP problem, as follows. The set of variables $\mathcal{X}$ and the set of domain values $\mathcal{D}$ are the same as defined before. The set of constraints is composed by the $0/\infty$-functions expressed as boolean functions (i.e., valuations 0 and ∞ are rewriten by $true$ and $false$, respectively), and the following constraints bounding the maximum accepted values for revenue loss and probability of payment failure,

$$F_1(\mathcal{X}) < P'$$

$$F_2(\mathcal{X}) < -\log R$$

where $P' = \sum_{i=1}^{m} m_i - P$.

B.2 Scheduling of an Earth Observation Satellite

Description

An earth observation satellite (EOS) orbits the earth while taking photographs requested by different customers. Each image i has two associated values: the penalty p_i for not taking it, resulting from the aggregation of several criteria like the client importance, the demand urgency and the meteorological forecasts, and the memory s_i required to store it in the on-board hard disk. The satellite has three on-board cameras. Each photograph can be

mono, that is, it can be taken by any one of the three cameras, or *stereo*, that is, it must be taken with the two most external cameras (i.e. front and rear cameras). Moreover, there exists a large number of imperative constraints such as non overlapping and sufficient transition time between successive images on the same instrument, limitation of the instantaneous data flow through the satellite telemetry resulting from simulataneous images on different instruments, etc. A very important constraint is the limitation of the on-board memory capacity for the images that are not directly down-linked. Typically, it is impossible to fulfil all the requests. Thus, the problem of scheduling the satellite consists in selecting the subset of photographs that the satellite will actually take minimizing the global aggregated penalty of discarded photographs. Clearly, it is a mono-objective optimization problem.

In the bi-objective version of the previous problem we would like to spent as less memory as possible. As a consequence, there are two objectives to optimize. The first one is to minimize the overall penalty of discarded photographs. The second one is to minimize the overall memory usage.

It it important to note that the on-board memory constraint in the mono-objective optimization problem is the second objective function in the bi-objective problem. As a consequence, the original problem can be solved as bi-objective. The solution of the mono-objective problem is the efficient solution of the bi-objective version such that the value of the second objective does not surpass the on-board available memory, and the cost of the first objective is minimum. Solving the original mono-objective instances as bi-objective may seem a bad option, since we are only concerned with one point of the efficient frontier. However, there is a very important structural property concerning the interaction graph of both formulations, as we will see in the encoding subsection.

Finally, there is a decision version of the above problem in which two constants (S, P) are given: the available on-board memory S, and the maximum acceptable aggregated penalty is P. Below those thresholds, we do not care whether one subset of photographs has a lesser penalty valuation nor a lesser

Inst. #	# of photographs	# of constraints	w^*
54	67	204	11
29	82	380	14
42	190	1204	26
28	230	4996	79
5	309	5312	39
404	100	610	19
408	200	2032	36
412	300	4048	36
414	364	9744	82
503	143	492	9
505	240	2002	22
507	311	5421	55
509	348	8276	86
1401	488	10476	93
1403	665	12952	93
1405	855	17404	93
1407	1057	20730	93
1502	209	203	5
1504	605	3583	20
1506	940	14301	67

Table B.2: Instances from the SPOT5 benchmark.

memory usage than another. The problem consists in finding a subset of photographs accomplishing these two constraints (as well as the imperative ones previously described).

Instances

We experiment on instances from the SPOT5 benchmark [12], which involves 20 instances (see Table B.2). These instances have been selected from 498 instances which have been built by a CNES (Centre National d'Études Spa-

tiales) simulator of the SPOT5 order book.

SPOT5 instances can be divided into *single* and *multiple* orbit. Single orbit instances, whose identification number is less than 1000, do not have any limitation to the on-board memory capacity. The others, whose identification number is greater than 1000, include the on-board memory capacity constraint. In all instances, the available memory is 200.

Instances 404, 408, 412, 414, 503, 505, 507, and 509 have been created from the same instance: the instance 414, which is the largest of all the instances without capacity constraint. To create the instances 404, 408 and 412 some images have been randomly removed. To create the instances 503, 505, 507, and 509 some images have been removed in order to limit the number of conflicts. Instances 54, 29, 42, 28, and 5 result from a selection among the previous ones.

Similarly, instances 1401, 1403, 1405, 1407, 1502, 1504, and 1506 have been created from the same instance: the instance 1407, which is the largest of all the instances with a recording capacity constraint. To create the instances 1401, 1403, and 1405 some images have been randomly removed. To create the instances 1502, 1504, and 1506 some images have been removed in order to limit the number of conflicts.

Single-orbit instances have been solved to optimality by a number of complete methods [13]. However, multiple-orbit instances are very challenging and remain unsolved (the only exception is instance 1502 where the capacity constraint is irrelevant because it can be trivially satisfied).

Since multiple-orbit instances are so hard to solve, in some experiments, we break those instances into subinstances (see Table B.3). Will refer to subproblems with the following notation: $X(i, j)$ denotes instance X restricted to the subset of consecutive variables $i \dots j$. When a subproblem coincides with a connected component of the overall problem (in the multi-objective sense as described in the next section), we will indicate it as $X(i, j)^*$. It is important to recall that all the multiple orbit instances have been derived from the largest instance 1407. As a result, it turns out that the same sub-

Inst. #	# of photos	# of constr.	w^*	b*			
				lex	min-fill	MWO	LEX-M
1504(0,183)*	184	1329	18	46	180	54	58
1504(184,206)*	23	112	7	8	19	8	7
1504(356,461)*	106	840	18	29	93	67	81
1504(462,508)*	47	301	10	21	30	20	13
1506(0,150)	151	1440	31	77	92	78	100
1506(151,228)	78	1107	24	71	43	69	34
1506(229,317)	89	1349	34	69	74	72	57
1506(679,761)*	83	1243	28	30	29	30	30
1405(762,854)*	93	2193	34	49	48	49	44
1407(0,147)	148	1442	29	79	85	80	97
1407(148,247)	100	1678	31	80	57	80	44
1407(248,378)	131	3063	52	103	116	112	83
1407(379,409)*	31	220	11	12	23	12	11
1407(413,429)*	17	87	8	9	15	9	8
1407(447,469)*	23	129	9	10	21	10	9
1407(494,553)*	60	1333	32	46	56	51	40
1407(580,700)	121	2299	44	86	116	86	93
1407(701,761)	61	445	13	28	60	30	42
1407(762,878)*	117	2708	34	49	48	49	47

Table B.3: Subinstances from the SPOT5 benchmark.

problem may have different names with our notation. When this is the case, we will name it as the subproblem coming from instance 1407.

Encoding

In one possible formulation of the mono-objective problem there is one variable for each photograph $x_1, \ldots, x_n$, and the domains of the variables are the values corresponding to the different alternatives for x_i:

- For *mono* photographs the domain is $\{0, 1, 2, 3\}$.

- For *stereo* photographs the domain is $\{0, 1\}$.

In both cases, domain value 0 indicates that the corresponding photograph is not taken. The other domain values correspond to the different alternatives to take the photograph.

Imperative constraints involve two and three photographs. They are encoded as binary and ternary $0/\infty$–functions $c_{ij}(x_i, x_j)$ and $c_{ijk}(x_i, x_j, x_k)$, respectively. Moreover, there is a unary function $f_i(x_i)$ for each variable x_i capturing the penalty p_i for not taking photograph i,

$$f_i(x_i) = \begin{cases} p_i & x_i = 0 \\ 0 & otherwise \end{cases}$$

Finally, the on-board memory capacity constraint is encoded as,

$$CC(\mathcal{X}) = \begin{cases} 0 & \sum_{i=1}^{n} w_i(x_i) < 201 \\ \infty & otherwise \end{cases}$$

where

$$w_i(x_i) = \begin{cases} 0 & x_i = 0 \\ s_i & otherwise \end{cases}$$

The objective function is $F(\mathcal{X}) = \sum f_i + \sum c_{ij} + \sum c_{ijk} + CC$. It is important to note that the capacity constraint involves all variables in the problem (i.e., it is an n–ary function). The interaction graph of this formulation is a clique. Namely, its induced width is the number of variables n. The main consequence is that the problem must be solved as a whole, and cannot be broken into independent parts.

The first objective function of the bi-objective version is the same as in the mono-objective case disregarding the capacity constraint. Namely, $F_1(\mathcal{X}) = \sum f_i + \sum c_{ij} + \sum c_{ijk}$. The second objective function is the capacity constraint. It can be encoded as a minimization WCSP problem, where the objective function is $F_2(\mathcal{X}) = \sum_{i=1}^{n} w_i$. As we have seen in Chapter 4, this bi-objective optimization problem can be encoded as a MO-WCSP problem $(\mathcal{X}, \mathcal{D}, \mathcal{F})$. The set of variables $\mathcal{X}$ and the set of domain values $\mathcal{D}$ is the same

as in the mono-objective case. The set of multi-objective cost functions $\mathcal{F}$ is composed by the previous unary, binary and ternary functions expressed as bi-objective cost functions. The set of unary bi-objective cost functions is,

$$\forall 1 \leq i \leq n,\ f'_i(x_i) = \begin{cases} \{(p_i, 0)\} & x_i = 0 \\ \{(0,0)\} & otherwise \end{cases}$$

$$\forall 1 \leq i \leq n,\ w'_i(x_i) = \begin{cases} \{(0,0)\} & x_i = 0 \\ \{(0, s_i)\} & otherwise \end{cases}$$

Note that the first and second set refer to the penalty paid when photograph i is not selected and to the storage memory spent when photograph i is selected, respectively. It is important to note that, with this formulation, multi-objective cost functions are at most ternary functions. The first consequence is that its induced width w^* can be smaller than n. The second consequence is that its interaction graph can be made of several independent parts. If that is the case, each part can be solved independently. The efficient frontier of the overall problem is the combination of the efficient frontier of each independent part.

Finally, the decision version of the problem can be modeled as a CSP problem $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X}$ and $\mathcal{D}$ are the same as in the mono-objective case. The set of constraints $\mathcal{C}$ is composed by the binary and ternary cost functions expressed as boolean functions. The bound over the maximum acceptable aggregated penalty P and on-board memory capacity S is encoded as the following constraints,

$$F_1(\mathcal{X}) < P$$

$$F_2(\mathcal{X}) < S$$

B.3 Max-SAT-ONE

Description

Consider a set of boolean variables $\{x_1, \ldots, x_n\}$ and the usual boolean operators $\wedge$, $\vee$ and $\neg$. A *literal* is either a variable (e.g., x_i) or its negation (e.g., $\neg x_i$). A *clause* $C = l_1 \vee \ldots \vee l_k$ is a disjunction of literals. A formula in conjunctive normal form is a conjunction of a number of clauses. A sample formula in conjunction normal form would be

$$(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2) \wedge (x_2 \vee \neg x_3)$$

Given a set of clauses $C_1, C_2, \ldots, C_m$ on the variables $x_1, x_2, \ldots, x_n$, the *satisfiability problem* (SAT) is to determine if the formula

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

is satisfiable. Namely, if there is an assignment of values to the variables so that the above formula evaluates to *true*. This problem is NP-complete [26].

When the formula is satisfiable, we can consider an optimization problem where the goal is to maximize the variables assigned to *true*. The *Max-ONE problem* [95] is to find an assignment to the variables so as to have all clauses C_j satisfied and the maximum number of variables is assigned to *true*.

Sometimes, when the formula cannot be satisfied, we are interested in the assignment that satisfies the maximum number of clauses. The *maximum satisfiability problem* (Max-SAT) [108] is to find an assignment of values to the variables so as to have the maximum number of C_j evaluating to *true*.

In our experiments we consider the simultaneous optimization of the number of satisfied clauses and the number of variables assigned to *true*. We call this problem *Max-SAT-ONE problem*.

Instances

We experiment with the well-known dimacs SAT instances [70] from the Second DIMACS Challenge. With current SAT solvers, these instances are

solved almost instantly for the SAT problem. However, they remain quite challenging for the Max-SAT-ONE problem.

File	Vars	Clauses	Sat?	w^*	b*			
					lex	min-fill	MWO	LEX-M
aim-50-1_6-no-1	50	80	No	15	49	45	45	35
aim-50-1_6-no-2	50	80	No	20	45	36	45	33
aim-50-1_6-no-3	50	80	No	19	45	37	49	28
aim-50-1_6-no-4	50	80	No	20	48	47	49	37
aim-50-1_6-yes1-1	50	80	Yes	18	49	47	46	32
aim-50-1_6-yes1-2	50	80	Yes	16	49	48	41	35
aim-50-1_6-yes1-3	50	80	Yes	19	48	45	47	32
aim-50-1_6-yes1-4	50	80	Yes	19	49	46	47	34
aim-50-2_0-no-1	50	100	No	26	47	45	48	38
aim-50-2_0-no-2	50	100	No	26	47	45	45	32
aim-50-2_0-no-3	50	100	No	26	46	43	45	40
aim-50-2_0-no-4	50	100	No	23	48	44	45	36
aim-50-2_0-yes1-1	50	100	Yes	25	48	47	49	38
aim-50-2_0-yes1-2	50	100	Yes	23	43	46	46	44
aim-50-2_0-yes1-3	50	100	Yes	23	46	47	46	45
aim-50-2_0-yes1-4	50	100	Yes	21	45	44	45	29
aim-50-3_4-yes1-1	50	170	Yes	32	48	47	48	42
aim-50-3_4-yes1-2	50	170	Yes	31	49	47	48	42
aim-50-3_4-yes1-3	50	170	Yes	31	49	48	49	39
aim-50-3_4-yes1-4	50	170	Yes	31	48	47	47	42
aim-50-6_0-yes1-1	50	300	Yes	37	49	44	49	39
aim-50-6_0-yes1-2	50	300	Yes	37	47	47	49	44
aim-50-6_0-yes1-3	50	300	Yes	37	48	48	48	43
aim-50-6_0-yes1-4	50	300	Yes	36	47	49	47	42
aim-100-1_6-no-1	100	160	No	40	98	83	98	84
aim-100-1_6-no-2	100	160	No	39	94	91	96	96
aim-100-1_6-no-3	100	160	No	40	96	93	97	71
aim-100-1_6-no-4	100	160	No	40	96	94	93	85

Continued on Next Page...

File	Vars	Clauses	Sat?	w^*	b*			
					lex	min-fill	MWO	LEX-M
aim-100-1_6-yes1-1	100	160	Yes	36	95	97	94	90
aim-100-1_6-yes1-2	100	160	Yes	37	99	92	91	65
aim-100-1_6-yes1-3	100	160	Yes	36	98	88	91	88
aim-100-1_6-yes1-4	100	160	Yes	36	95	92	90	73
aim-100-2_0-no-1	100	200	No	52	97	92	98	89
aim-100-2_0-no-2	100	200	No	52	98	92	99	82
aim-100-2_0-no-3	100	200	No	50	95	95	96	70
aim-100-2_0-no-4	100	200	No	51	98	97	95	85
aim-100-2_0-yes1-1	100	200	Yes	45	96	99	95	76
aim-100-2_0-yes1-2	100	200	Yes	42	95	94	93	83
aim-100-2_0-yes1-3	100	200	Yes	47	96	92	92	77
aim-100-2_0-yes1-4	100	200	Yes	44	98	98	97	60
aim-100-3_4-yes1-1	100	340	Yes	62	95	91	92	78
aim-100-3_4-yes1-2	100	340	Yes	61	99	97	98	75
aim-100-3_4-yes1-3	100	340	Yes	64	95	98	98	79
aim-100-3_4-yes1-4	100	340	Yes	64	98	96	95	82
aim-100-6_0-yes1-1	100	600	Yes	72	97	95	99	84
aim-100-6_0-yes1-2	100	600	Yes	73	98	98	97	93
aim-100-6_0-yes1-3	100	600	Yes	76	98	97	98	90
aim-100-6_0-yes1-4	100	600	Yes	72	99	98	98	89
aim-200-1_6-no-1	200	320	No	90	194	192	197	156
aim-200-1_6-no-2	200	320	No	84	186	186	195	174
aim-200-1_6-no-3	200	320	No	82	198	189	197	190
aim-200-1_6-no-4	200	320	No	85	194	190	193	171
aim-200-1_6-yes1-1	200	320	Yes	72	197	193	187	158
aim-200-1_6-yes1-2	200	320	Yes	68	198	185	195	146
aim-200-1_6-yes1-3	200	320	Yes	68	194	195	188	153
aim-200-1_6-yes1-4	200	320	Yes	74	197	195	196	151
aim-200-2_0-no-1	200	400	No	97	195	194	192	173
aim-200-2_0-no-2	200	400	No	103	193	187	192	184
aim-200-2_0-no-3	200	400	No	103	194	197	190	179

Continued on Next Page...

File	Vars	Clauses	Sat?	w^*	b*			
					lex	min-fill	MWO	LEX-M
aim-200-2_0-no-4	200	400	No	104	197	197	198	164
aim-200-2_0-yes1-1	200	400	Yes	91	198	194	190	163
aim-200-2_0-yes1-2	200	400	Yes	93	197	191	197	134
aim-200-2_0-yes1-3	200	400	Yes	92	194	195	191	134
aim-200-2_0-yes1-4	200	400	Yes	92	198	195	194	173
aim-200-3_4-yes1-1	200	680	Yes	129	193	191	195	170
aim-200-3_4-yes1-2	200	680	Yes	125	194	196	189	161
aim-200-3_4-yes1-3	200	680	Yes	131	196	191	196	156
aim-200-3_4-yes1-4	200	680	Yes	127	199	193	199	159
aim-200-6_0-yes1-1	200	1200	Yes	155	197	194	195	189
aim-200-6_0-yes1-2	200	1200	Yes	155	197	194	196	172
aim-200-6_0-yes1-3	200	1200	Yes	155	198	194	196	178
aim-200-6_0-yes1-4	200	1200	Yes	153	198	198	192	178
dubois100	300	800	No	3	201	3	200	4
dubois20	60	160	No	3	41	3	40	4
dubois21	63	168	No	3	43	3	42	4
dubois22	66	176	No	3	45	3	44	4
dubois23	69	184	No	3	47	3	46	4
dubois24	72	192	No	3	49	3	48	4
dubois25	75	200	No	3	51	3	50	4
dubois26	78	208	No	3	53	3	52	4
dubois27	81	216	No	3	55	3	54	4
dubois28	84	224	No	3	57	3	56	4
dubois29	87	232	No	3	59	3	58	4
dubois30	90	240	No	3	61	3	60	4
dubois50	150	400	No	3	101	3	100	4
pret150_25	150	400	No	4	138	133	138	83
pret150_40	150	400	No	4	138	133	138	83
pret150_60	150	400	No	4	138	133	138	83
pret150_75	150	400	No	4	138	133	138	83
pret60_25	60	160	No	4	56	54	56	11

Continued on Next Page...

File	Vars	Clauses	Sat?	w^*	b*			
					lex	min-fill	MWO	LEX-M
pret60_40	60	160	No	4	56	54	56	11
pret60_60	60	160	No	4	56	54	56	11
pret60_75	60	160	No	4	56	54	56	11
ssa0432-003	435	1027	No	18	405	411	405	359
ssa2670-130	1359	3321		24	1275	1244	1275	1241
ssa2670-141	986	2315	No	21	951	870	951	804
ssa7552-038	1501	3575		29	1442	1474	1442	1178
ssa7552-158	1363	3034	Yes	11	1311	1352	1311	1034
ssa7552-159	1363	3032	Yes	11	1311	1352	1311	1034
ssa7552-160	1391	3126	Yes	14	1336	1332	1336	1068
hole6	42	133	No	26	36	40	36	36
hole7	56	204	No	35	49	53	49	49
hole8	72	297	No	45	64	67	64	64
hole9	90	415	No	57	81	83	81	81
hole10	110	561	No	71	100	103	100	100
jnh1	100	850	Yes	94	99	97	99	98
jnh2	100	850	No	94	99	95	99	97
jnh3	100	850	No	94	99	96	99	97
jnh4	100	850	No	92	99	98	98	97
jnh5	100	850	No	93	99	94	99	95
jnh6	100	850	No	93	99	93	98	98
jnh7	100	850	Yes	94	99	96	99	96
jnh8	100	850	No	93	99	96	99	97
jnh9	100	850	No	93	99	98	98	97
jnh10	100	850	No	94	99	96	99	96
jnh11	100	850	No	93	99	95	99	96
jnh12	100	850	Yes	93	99	99	99	96
jnh13	100	850	No	93	99	96	98	95
jnh14	100	850	No		99	97	99	98
jnh15	100	850	No	93	99	96	99	95
jnh16	100	850	No	93	99	97	99	96

Continued on Next Page...

File	Vars	Clauses	Sat?	w^*	b*			
					lex	min-fill	MWO	LEX-M
jnh17	100	850	Yes	94	99	97	98	96
jnh18	100	850	No	94	99	96	99	96
jnh19	100	850	No	93	99	98	99	96
jnh20	100	850	No	93	99	94	99	97
jnh201	100	800	Yes	94	99	95	99	96
jnh202	100	800	No	93	99	93	98	96
jnh203	100	800	No	92	99	98	99	96
jnh204	100	800	Yes	93	99	98	98	96
jnh205	100	800	Yes	92	99	97	97	95
jnh206	100	800	No	93	99	99	99	96
jnh207	100	800	Yes	93	99	97	99	95
jnh208	100	800	No	93	99	98	99	96
jnh209	100	800	Yes	93	99	98	99	96
jnh210	100	800	Yes	93	99	95	98	96
jnh211	100	800	No	92	99	97	98	96
jnh212	100	800	Yes	92	99	95	99	99
jnh213	100	800	Yes	94	98	96	97	97
jnh214	100	800	No	92	99	93	99	96
jnh215	100	800	No	93	99	99	99	97
jnh216	100	800	No	92	99	96	99	96
jnh217	100	800	Yes	92	99	98	98	96
jnh218	100	800	Yes	93	99	95	99	96
jnh219	100	800	No	93	99	97	99	95
jnh220	100	800	Yes	92	99	95	99	97
jnh301	100	900	Yes	94	99	96	99	97
jnh302	100	900	No	94	99	96	99	95
jnh303	100	900	No	93	99	96	99	97
jnh304	100	900	No	94	99	94	98	98
jnh305	100	900	No	94	99	99	99	96
jnh306	100	900	No	93	99	94	99	98
jnh307	100	900	No	94	99	98	98	97

Continued on Next Page...

File	Vars	Clauses	Sat?	w^*	b*			
					lex	min-fill	MWO	LEX-M
jnh308	100	900	No	94	99	98	98	96
jnh309	100	900	No	93	99	96	98	97
jnh310	100	900	No	94	99	96	97	97

Table B.4: Dimacs sat benchmark.

Details on the instances are given in Table B.4. The first letters of the name of the instances identify its source, that is, the type of problem encoded and its generator. In the following, we outline these sources:

SSA (from Allen Van Gelder and Yumi Tsuji) Instances from circuit fault analysis: checking for circuit "single–stuck–at" fault. The instances are selected formulas from those generated by Nemesis, a test-pattern generation program described in [45, 85].

Dub (from Olivier Dubois) Instances randomly generated. The generator of the instances (gensathard.c) is available at the DIMACS ftp-site[3]. All the instances are unsatisfiable.

Pret (from Daniele Pretolani) An encoding of two–coloring a graph, along with a parity constraint to force unsatisfiablity. The generator (trisat.c) is publicly available[4].

JNH (from John Hooker) Random instances generated in the following way: For an instance with n variables and k clauses, clauses are generated by including a variable with a fixed probability p, and then negating the variable with probability 0.5. Formulas generated in this way may contain empty clauses or unit clauses. Hence, empty clauses and unit clauses are rejected in the generation process. The resulting problem distribution is called Random P-SAT in [132].

[3]ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSC/instances
[4]ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/pretolani

Hole (from John Hooker) Instance of the pigeon hole problem. The Pigeon Hole problem asks whether it is possible to place n+1 pigeons in n holes without two pigeons being in the same hole.

AIM (from Eiji Miyano) Artificially generated 3–sat instances. All the "yes" instances have exactly one satisfying assignment. The generators are descibed in [5] and are publicly available[5].

Encoding

We used the following encoding of Max-SAT-ONE instances to MO-WCSP. First, it is important to recall that we are considering the minimization of both objective functions. Then, given a formula $C_1 \wedge \ldots \wedge C_m$ in conjunctive normal form with n variables $x_1, \ldots, x_n$, the corresponding MO-WCSP problem $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ is as follows. There is a variable in $\mathcal{X}$ for each variable in the formula and its domain values are $true$ and $false$. For each clause C_j, we have a multi-objective cost function that penalize the assignments of variables in C_j that do not satisfy it. Formally,

$$\forall\, 1 \leq j \leq m,\ c_j(var(C_j)) = \begin{cases} \{(1,0)\} & \neg C_j \\ \{(0,0)\} & otherwise \end{cases}$$

where $var(C_j)$ is the set of variables (either negated or not) that appear in the clause C_j. Moreover, there is a set of unary multi-objective cost functions that penalize the $false$ assignment to any variable. Formally,

$$\forall\, 1 \leq i \leq n,\ p_i(x_i) = \begin{cases} \{(0,1)\} & \neg x_i \\ \{(0,0)\} & otherwise \end{cases}$$

Note that functions c_j refer to the first objective function while functions p_i refer to the second objective function.

[5] ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/iwama/

B.4 Biobjective Weighted Minimim Vertex Cover

Description

Consider a graph $G = (V, E)$ where V is the set of vertexs and E is the set of edges. A *vertex cover* (or *hitting set*) is a subset S of vertices such that each edge of G has at least one of its endpoints in S. Formally,

$$S \subseteq V \text{ such that } \forall (u, v) \in E, \text{ either } u \in S \text{ or } v \in S$$

The *minimum vertex cover problem* is to find a vertex cover with the smallest number of vertices. This problem is NP-complete [73].

Consider that each vertex $u \in V$ has an associated weight $w(u)$. Then, the *weighted minimum vertex cover problem* is to find a vertex cover with minimum aggregated weight.

The problem considered in our experiments is a bi-objective extension of the previous weighted problem. Each vertex $u \in V$ has two associated weights $w_1(u)$ and $w_2(u)$. The *bi-objective weighted minimum vertex cover problem* is to find the vertex cover minimizing at the same time the aggregated weight w_1 and w_2.

Instances

We generated instances for the bi-objective weighted minimum vertex cover as follows. First, we generated random graphs with N vertexs and E edges. Then, for each vertex, two costs are randomly generated from the interval $[0 \dots C]$. It is important to note that the values of N, E and C determine different classes of instances. For each parameter configuration (N, E, C) we generated samples of size 25.

We experiments on samples of the following classes of problems

$$(\{60, 70, 80, 90\}, \{100, 250, 500, 950\}, 5)$$

See Figure B.5 for details on these instances.

V	E	w^*	b*			
			lex	min-fill	MWO	LEX-M
60		9.5	55.2	51.9	53.8	45.6
70	100	8.5	64.4	60.2	61.8	54.3
80		7.3	72.5	72.0	69.4	57.5
90		6.1	79.6	80.8	78.5	68.8
60		27.0	56.9	54.8	56.4	47.3
70	250	26.9	66.3	63.1	65.3	58.2
80		27.3	75.8	72.7	73.9	66.2
90		29.2	84.6	80.0	83.5	73.3
60		39.9	57.6	56.0	57.3	49.8
70	500	41.3	67.6	65.9	66.3	58.9
80		45.6	76.7	75.0	76.0	66.6
90		47.0	87.1	84.8	85.2	73.9
60		48.3	58.3	57.0	58.3	53.4
70	950	53.7	68.0	67.1	68.0	61.8
80		57.7	77.5	76.2	77.3	69.1
90		63.2	87.3	86.5	86.7	78.0

Table B.5: Mean value of the induced width and bandwidth for different variable orderings of the bi-objective weighted vertex cover on 25 instances.

Encoding

A biobjective minimum vertex cover can be expressed as a MO-WCSP $(\mathcal{X}, \mathcal{D}, \mathcal{F})$, as follows. Given a graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$, there is a variable x_i for each vertex v_i (i.e., $\mathcal{X} = \{x_1, \ldots, x_n\}$). The domains of the variables are the values 1 and 0:

- domain value 1 represents that vertex x_i is contained in the vertex cover; and

- domain value 0 represents that vertex x_i is not part of the vertex cover.

The subset of variables assigned to domain value 1 must be a vertex cover. This condition can be expressed as a set of binary constraints,

$$\forall (v_i, v_j) \in E,\ c_{ij}(x_i, x_j) = x_i \vee x_j$$

Since we are in a multi-objective context, these constraints are expressed as multi-objective cost functions,

$$\forall (v_i, v_j) \in E,\ c_{ij}(x_i, x_j) = \begin{cases} \{(0,0)\} & x_i \vee x_j \\ \{(\infty,\infty)\} & otherwise \end{cases}$$

Finally, the costs $w_1(v_i)$ and $w_2(v_i)$ associated to each vertex v_i can be expressed as a set of unary multi-objective cost functions,

$$\forall\, v_i \in V,\ f_i(x_i) = \begin{cases} \{(w_1(v_i), w_2(v_i))\} & x_i = 1 \\ \{(0,0)\} & x_i = 0 \end{cases}$$

Note that f_i specifies that it is preferred not to add vertex x_i to the vertex covering.

B.5 Maxclique

Description

Consider a graph $G = (V, E)$ where V is the set of vertexs and E is the set of edges. A *clique* is a subset $C \subseteq V$ such that each pair of vertices in S is connected by an edge in E. Formally,

$$C \subseteq V \text{ such that } \forall\, u, v \in C\ (u, v) \in E$$

The *maximum clique problem* is to find a clique with the largest number of vertices in a given graph. This problem is NP-complete [73].

As noted in [44], finding the maximum clique of a graph $G = (V, E)$ is equivalent to finding a minimum vertex cover of the complementary graph $\overline{G}$. Given a graph $G = (V, E)$, its complementary graph is denoted by $\overline{G} =$

$(V, \overline{E})$. It is constructed with the same set of vertices V and $(v_i, v_j) \in \overline{E}$ iff $(v_i, v_j) \notin E$. Hence, we can solve a maxclique problem as a minimum vertex cover problem over the complementary graph. Observe that the maximum size of the maximum clique is equivalent to $|V - S|$, where S is the minimum vertex cover.

Instances

We considered instances from the Second DIMACS Challenge [70]. This benchmark is only used in Appendix A.2. Figures A.10 and A.11 report the important features of these instances. Each instance comes from a source. Each source is based in a particular problem. It is easy to identify from the name of each instance its source. We outline these problems in the following:

CFat (from Panos Pardalos) Problems based on fault diagnosis problems.

Joh (from Panos Pardalos) Problems based on problems in coding theory. A Johnson graph with parameters n, w, and d has a node for every binary vector of length n with exactly w 1's. Two vertices are adjacent if and only if their hamming distance is at least d. A clique then represents a feasible set of vectors for a code.

Kel (from Peter Shor). Problems based on Keller's conjecture on tiling using hypercubes.

Ham (from Panos Pardalos). Another coding theory problem. A Hamming graph with parameters n and d has a node for each binary vector of length n. Two nodes are adjacent if and only if the corresponding bit vectors are hamming distance at least d apart.

San (from Lausa Sanchis) Instances based on her "Test Case Construction for Vertex Cover Problem", DIMACS workshop on Computational Support for Discrete Mathematics, March 1992. The generator generates instances with known clique size.

Bro (from Patrick Soriano and Michel Gendreau) Random problems generated with the *p_hat* generator [55] wich is a generalization of the classical uniform random graph generator. It uses 3 parameters: n, the number of nodes, and a and b, two density parameters verifying $0 \leq a \leq b \leq 1$.

Stein (from Carlo Mannino) Clique formulation of the set covering formulation of the Steiner Triple Problem [77]. Created using Mannino's code to convert set covering problems to clique problems.

Encoding

As we have seen, a maxlique problem can be translated into a minimum vertex cover problem over the complimentary graph. Following this equivalence, we encode a maxclique problem as a WCSP $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ as follows.

Given a graph $G = (V, E)$ (with $|V| = n$) there is a variable for each vertex $\{x_1, \ldots, x_n\}$. The domains of the variables are the values 1 and 0:

- domain value 1 represents that vertex x_i is not part of the clique (and it is part of the vertex cover); and

- domain value 0 represents that vertex x_i is contained in the clique (and it is not contained in the vertex cover).

Then, the condition impossed for the variables taking domain value 1 to be a vertex cover is,

$$\forall (v_i, v_j) \notin E,\ c_{ij}(x_i, x_j) = x_i \vee x_j$$

Note that when a variable x_i takes domain value 0 the only variables allowed to take domain value 0 are the ones representing a vertex connected with v_i (i.e., the ones that can be part of a clique in G). Finally, there is a set of unary cost functions in order to specify that is preferred not to add vertexs to the vertex cover (conversally, that is preferred to add vertexs to the clique),

$$\forall\, v_i \in E,\ f_i(x_i) = x_i$$

B.6 Most Probable Explanation

Description

As we have seen in Section 2.3, belief networks are a well-known graphical model that allows us to reason with probabilities. One important reasoning task posed on a belief network is the *most probable explanation* (MPE), where the objective is to find the assignment with maximum probability distribution.

Instances

In our experiments, we generated instances for the MPE on two types of belief networks: *uniform random* and *noisy-OR networks* [74].

Both networks are generated using parameters (N, K, C, P), where N is the number of variables, K is their domain size, C is the number of conditional probability tables (CPT), and P is the number of parents in each CPT. Instances are generated by selecting C variables at random. For each selected variable x_i, P parents are randomly selected from the set of variables with index less than i (if $i \leq P$ only $i-1$ parents are selected). For each parameter setting we generate a sample of 20 instances.

For random bayesian networks, each CPT is randomly generated using a uniform distribution. For noisy-OR networks, each CPT represents a noisy OR-function. For each CPT, we randomly assign to each parent variable y_j a value $P_j \in [0 \dots P_{noise}]$. The CPT is then defined as, $P(x = 0|y_1, \dots, y_P) = \prod_{y_j=1} P_j$ and $P(x = 1|y_1, \dots, y_P) = 1 - P(x = 0|y_1, \dots, y_P)$.

Figure B.2 shows the important details of the instances generated.

Encoding

It is easy to see that the MPE problem can be expressed as a WCSP problem by replacing probability tables by their logarithm. Given a MPE problem $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ where the objective function is to maximize $\prod_{f \in \mathcal{F}} f$, its equivalent

N, C, P	w^*
Uniform Random Bayesian Networks	
128, 85, 4	31.71
128, 95, 4	43.96
128, 105, 4	38.71
128, 115, 4	48.32
Noisy-OR $P_{noise} = 0.40$	
128, 85, 4	35.39
128, 95, 4	38.61
128, 105, 4	43.06
128, 115, 4	46.51
Noisy-OR $P_{noise} = 0.50$	
128, 85, 4	40.74
128, 95, 4	38.12
128, 105, 4	43.04
128, 115, 4	46.25

Figure B.2: Most Probable Explanation benchmark. 20 samples.

WCSP problem is $(\mathcal{X}, \mathcal{D}, \mathcal{F}')$ where $\mathcal{F}'$ is defined as,

$$\forall\, f \in \mathcal{F},\ \forall\, t \text{ such that } var(t) = var(f),\ f'(t) = -\log f(t)$$

The objective function is to minimize $\sum_{f' \in \mathcal{F}'} f'$.

B.7 Frequency Assignment

Description

Consider a radio communication network, defined by a set of radio links. When radio communication links are assigned the same or closely related frequencies, there is a potential for interference. The *radio link frequency assingment problem* (RLFAP) is to assign, from limited spectral resources, a

frequency to each of these links in such a way that all the links may operate together without noticeable interference. Moreover, the assignment has to comply to certain regulations and physical constraints of the transmitters.

Once there exists an assignment accomplishing the regulations and physical constraints, we may want to make good use of the available spectrum, trying to save the spectral resources for a later extension of the network. Then, several optimization problems arise. For example, minimizing the largest frequency used in the assignments (called *minimum span*), minimizing the number of different frequencies used (called *minimum cardinality*), or minimizing the sum of all interference costs (called *maximum feasibility*). We focus in the maximum feasibility RLFAP problem.

Initially, frequency assignment problems were proposed by the French Center d'Electronique de l'Armament (CELAR) in the framework of the European EUCLID project CALMA (Combinatorial Algorithms for Military Applications). Within this project, several techniques, mainly from Operations Research, have been applied to frequency assignment problems (see [140] for a detailed description). These techniques include branch and cut, constraint satisfaction, local search, genetic algorithms, or potential reduction. Most of these techniques proved to be efficient for minimum span and minimum cardinality problems, but not for maximum feasibility ones. In the CSP/CLP community, these instances have been used for assessing the performance of arc-consistency enforcing algorithms, for satisfaction algorithms or for the computation of lower bounds in constraint optimization problems (as it is also our case).

Instances

There are two types of RLFAP instances called CELAR and GRAPH instances. The first ones were proposed as a simplified versions of a real-world instance comming from the telecommunication industry. The latters were generated with the GRAPH generator (Generating Radio link frequency Assignment Technology Heuristically, [14]) during the CALMA project. We

Inst. #	# of var.	# of. constr.	w^*
05	200	1134	135
06	400	2170	296
07	400	2170	146
11	680	3757	495
12	680	4017	234
13	916	5273	706

Table B.6: Maximum feasibility GRAPH instances.

experiment on GRAPH instances. Table B.6 summarize its main properties of the maximum feasibility instances. The first column indicates the instance number. The second and third columns indicate the number of variables and the number of constraints, respectively. The last column shows the induced width given by the min-fill order.

These instances were also proposed in the CSP community as a benchmark [23]. The reason is that they can be described in terms of graphical models as a CSP, probabilistic and WCSP problems. In particular, maximum feasibility is a WCSP problem, where all the constraints are binary and the domain values are discrete and finite, as we will see in the following.

Encoding

The maximum feasibility RLFAP problem can be modeled as a WCSP problem. Each radio link is represented by a variable. For each link i, a frequency f_i has to be chosen from a finite set D_i of frequencies available for the transmitter. There are binary constraints between two links i and j that imposses a minimum distance between the frequencies f_i and f_j assigned to variables x_i and x_j, respectively. If that distance is not maintained, then there exists an interference cost c_i. These requirements are modelled as,

$$c_{ij}(x_i, x_j) = \begin{cases} 0 & |f_i - f_j| > d_{ij} \\ p_i & otherwise \end{cases}$$

where d_{ij} is the minimum distance which must separate two frequencies and p_i is the cost for violating the distance constraint. Similarly, there is a technologic binary constraint which imposses a distance of 238 for duplex frequences.