Solving Multi-Objective Distributed Constraint Optimization Problems

CLEMENT Maxime

Doctor of Philosophy

Department of Informatics
School of Multidisciplinary Sciences
SOKENDAI (The Graduate University for Advanced Studies)

DOCTORAL THESIS

Solving Multi-Objective Distributed Constraint Optimization Problems

Author:
CLEMENT Maxime

Supervisor:
Prof. INOUE Katsumi

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies

July 2017

Abstract

For many decades, the field of optimization has progressed both in the way it represents real-life problem and how it solves these problems. However, most of the focus have been towards problems with a single objective to optimize whereas almost all real-life problems involve multiple objectives.

In this thesis, we study the solving of multi-objective problems that we propose to view as two successive steps. The first step consists in finding the Pareto front of the given problem, i.e., the set of solutions offering optimal trade-offs of the objectives. As the size of this Pareto front can increase exponentially with the size of the problem, efficient algorithms are required to find the Pareto front or at least a good approximation. This first step is thus mostly about algorithms and searching for solutions. The second step then consists in using the previously found Pareto front to extract a single solution that can be implemented. It can be assumed that this second step is performed by a decision maker. However, selecting a solution from the Pareto front is rarely trivial for a human as the set of alternatives can be very large. Additionally, many problems require quick decisions, in which case interacting with a decision maker can be impossible. This second step is more about decision making and defining additional criteria to extract solutions from the Pareto front.

In the first chapter of this thesis, we present the background of the main fields of research to which our thesis contributes. These fields are Optimization, Constraint Programming and Distributed Problem Solving.

In the second chapter, we introduce the preliminary notions required for the understanding of our contribution. We present the framework used to represent multi-objective problems both for centralized and distributed systems, as well as for dynamic environments where the problem changes over time. In addition, we present some of the representative applications considered during this thesis.

In the third chapter, we study the first step of multi-objective problem solving and propose two new algorithms for finding the Pareto front of multi-objective optimization problems in distributed systems. The first algorithm is the Multi-Objective Distributed Pseudo-tree Optimization, an exact algorithm based on dynamic programming techniques to find the complete Pareto front. The second algorithm is the Distributed Pareto Local Search, an algorithm based on local search techniques that provides an approximation of the Pareto front.

In the fourth chapter, we study the second step of multi-objective problem solving and propose three multi-objective decision making methods to isolate subsets of Pareto fronts that can be deemed more interesting. The first selection method uses a preference-based criterion while the second and third selection methods use criteria specific to dynamic problems.

In the fifth chapter, we discuss the related works and compare them with our proposed algorithms and decision methods.

In the final chapter, we conclude this thesis by summing up our contribution and discussing the potential future works related to this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Most decisions made in life rely on some form of optimization process where different decisions are considered but only one is implemented in the end. Automated optimization is a crucial part of a wide variety of systems, often greatly improving their efficiency and reducing their costs. An example is how the optimization of schedules for nurses in hospitals or for teachers in universities greatly improves the quality of the services provided. Another example is how the optimization of the routes taken by delivery trucks reduces the time taken to perform all deliveries.

Many such optimization problems require the consideration of multiple objectives that should be considered separately but optimized simultaneously. Objectives should be considered separatly when there is no good way to express one objective in relation to the other objectives. For example, when dealing with the security and speed of a network, we cannot express the security as a factor of the speed. While the objectives are considered separatly, they should be optimized simultaneously as we want to obtain a solution achieving the best value for each objective without ignoring any of them. In practice, the different objectives are usually concurrent to eachother such that improving one of the objectives will decrease some other objectives. Thus, multi-objective problems rarely accept a single solution that is the best for all objectives simultaneously. Instead, there exists several solutions that offer different trade-offs of the objectives. This set of solutions is called the Pareto front and selecting any solution from this set guarantees that there will not exist another solution that is better for all objectives.

Figure 1.1 shows the full optimization process both in the mono-objective case and the multi-objective case. We can see that in the mono-objective case, a solution can directly be implemented after the problem was solved whereas in the multi-objective case, an extra decision must be made after finding the Pareto front. The mono-objective case can actually be seen as a special case of optimization where the Pareto front will always contain a single solution, allowing us to skip the extra decision step.

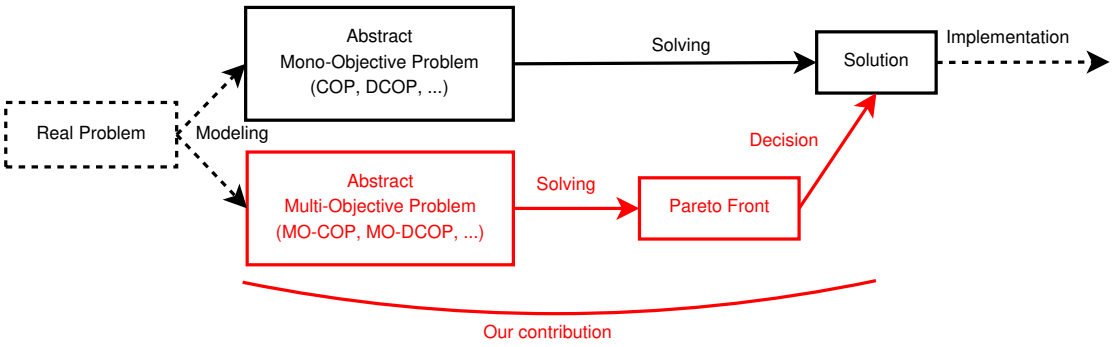


FIGURE 1.1: Representation of the typical optimization process, from the abstraction of the real problem to the implementation of the solution found.

There have been many works on multi-objective optimization and there now exists an extensive range of techniques to find or approximate the Pareto front of large multi-objective problems. The final decision step is usually assumed to be performed by interacting with a decision maker and little research has been done about automating this selection step. In this thesis, we study problems that are represented using constraints using techniques from constraint programming.

Constraint programming is a field of artificial intelligence (AI) where problems are defined using a set of variables and a set of constraints between these variables. Variables are usually assumed to have a finite domain and a constraint defines a relation between the values of its variables. Constraint programming divides the modeling and solving of problems by providing declarative models and general solvers for these models. There exists a multitude of languages and frameworks to both model and solve constraint problems. Representing optimization problems using constraints offer two interesting aspects. First, it can naturally represent relations between different entities of a real world problem. For example, two meetings correspond to two different variables naturally share a constraint if they should not be scheduled at the same time. Secondly, using constraints offers a decomposition of the problem into independent parts, which can be exploited to design efficient algorithms. These two points made constraint optimization frameworks very attractive to the multi-agent community as they can easily model various multi-agent coordination problems.

The study of multi-agent systems (MAS) is a field of AI that is becomming increasingly important with the rise of distributed and integrated systems. Whereas AI usually assumes a single rational agent, a multi-agent system (MAS) considers a set of autonomous agents interacting with eachother. When these agents work together to solve a problem, we talk about coordination, i.e., each agent will take decisions that benefit the whole group of agents.

Distributed optimization has been extensively studied in the last two decades and many algorithms have been proposed, allowing to tackle a number of real problems in various situations. However, distributed multi-objective optimization is still a relatively new field.

This thesis contributes to the solving of multi-objective optimization problems by developing new algorithms that compute the Pareto front in a distributed fashion and by proposing new methods to select the solution to implement from a Pareto front. These contributions corresponding to the solving and decision making part of the optimization process, highlighted in red in Figure 1.1. We assume that we are given an abstract multi-objective problem using the framework called Multi-Objective Constraint Optimization Problem (MO-COP) or its variants for multi-agent systems called Multi-Objective Distributed Constraint Optimization Problem (MO-DCOP).

While distributed optimization is well studied in the mono-objective case, its multi-objective extension is still very new with very few techniques available to find the Pareto front of distributed problems. We contribute to this field by proposing two new algorithms that uses different techniques suited for different case. A first algorithm that guarantees to find the exact Pareto front uses dynamic programming techniques often used in constraint programming. The second algorithm finds an approximation of the Pareto front using local search techniques that are widely used in optimization.

After the Pareto front is found, we need to select a solution to implement. We consider in this thesis some situations where the selection step cannot be directly performed by a decision maker, either because the Pareto front is too difficult to comprehend or because interacting with a decision maker is inconvenient. For example, we will consider dynamic scenarios where rapid decisions must be made, requiring to automate the decision as interacting with the decision maker would take too much time.

1.1 Background

In this section we present the background of fields our contribution belongs to, namely: Optimization in section 1.1.1, Constraint Programming in section 1.1.2, and Distributed Problem Solving in section 1.1.3.

1.1.1 Optimization

Optimization [1] is a branch of mathematics looking to model, analyze and solve problems consisting in minimizing or maximizing an objective function. Optimization is a key part of both artificial intelligence and operational research where many fundamental problems (pathfinding, resource allocation, binpacking, ...) can be represented as optimization problems. Optimization problems can be classified using many different properties. When a problem includes the consideration of some constraints, it is called a constrained optimization problem. Otherwise, it is called an unconstrained optimization problem. When the objective functions to optimize (as well as the constraints) are linear functions, the problem is called a linear programming problem. There also exists geometric programming problems and quadratic programming problems. When variables take only discrete (or integer) values, the problem is called an integer programming problem. When the values are real, the problem is called a real-valued programming problem. Based on the number of objective functions to optimize, we can say that a problem is a mono-objective problem (or single-objective problem) if it has only one objective function.

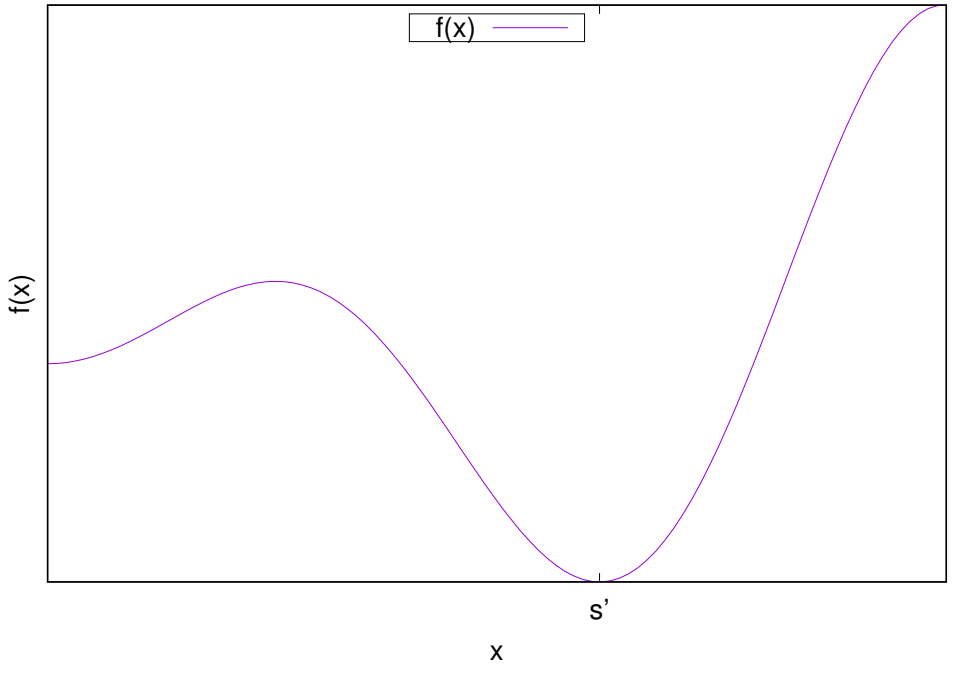


FIGURE 1.2: Example of objective function $f(x)$ and its optimal solution s'.

If it has more than one objective function, the problem is said to be a multi-objective problem. In this thesis, the optimization problems considered are constrained, non-linear, discrete, and multi-objective.

Generally speaking, an optimization problem considers:

- N is the solution space.
- $f : N \to \mathbb{R}$ is an objective function defining a value for each element of the solution space N.

Then, in a minimization problem, the goal is to find an element $s' \in N$ such that $f(s') \leq f(s)$ for all $s \in N$, also written:

$$s' = \underset{s \in N}{\operatorname{argmin}} f(s)$$

Example 1.1 (Optimization)**.** *Figure 1.2 shows a simple example of a continuous objective function $f(x)$ and its optimal solution s'.*

Next, we will briefly introduce the types of optimization considered in this thesis, namely *combinatorial* and *multi-objective* optimization.

1.1.1.1 Combinatorial Optimization

Combinatorial optimization [2] is a branch of optimization where the solution space, i.e., the set of possible solutions, is discrete and finite.

Formally, we consider a combinatorial optimization problem where:

- N is a *discrete* set of solution.
- $f : 2^N \to \mathbb{R}$ is an objective function defining a value for each combination of elements in N.
- $\mathcal{F} \subseteq 2^N$ is a set of subsets of N listing all *feasible* solutions of the problem.

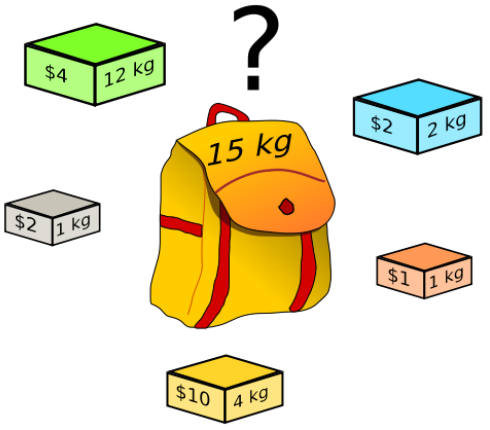
Then, a minimization combinatorial problem can be written

$$s' = \underset{s \in N}{\operatorname{argmin}} f(s) \; s.t. \; s \in \mathcal{F}$$

Example 1.2 (Knapsack Problem)**.** *A classical combinatorial optimization problem is the knapsack problem [3, 4] where we are given a set of items $N = \{n_1, \ldots\}$ where each item n_i have a weight w_i and a value v_i, and we want to find a subset of items whose total weight*

is less or equal a given limit and whose total value should be maximized. The figure shows an example of knapsack problem where the weight limit is 15kg and 5 items are available:

- n_{red} , w_{red} $= 1$, v_{red} $= 1$.
- n_{grey} , w_{grey} $= 1$, v_{grey} $= 2$.
- n_{blue} , w_{blue} $= 2$, v_{blue} $= 2$.
- n_{yellow}, w_{yellow} $= 4$, v_{yellow} $= 10$.
- n_{green} , w_{green} $= 12$, v_{green} $= 4$.



[1]

The solution space of this problem is the set of combinations of items.The feasible solutions are the combinations of items whose total weight is less than or equal to 15, i.e., $\sum_{s_i \in S} w_i \leq 15$, $S \in 2^N$. *For example,* $\{Blue, Yellow, Green\}$ *is not a feasible solution since its weight is higher than 15 (*$2+10+4=16$*).* $\{Red, Yellow, Green\}$ *is a feasible solution since its weight is not higher than 15 (*$1+10+4=15$*).*

The objective function can be written as $f(S) = \sum_{s_i \in S} v_i$. *The optimal solution of this problem is* $\{Red, Grey, Blue, Yellow\}$ *with a weight of* $8(1+1+2+4)$ *and a value of* $15(1+2+2+10)$.

1.1.1.2 Multi-Objective Optimization

When multiple objectives should be optimized simultaneously, it is called Multi-Objective Optimization (MOOP) [5]. MOOP is a branch of combinatorial optimization whose specificity is to simultaneously attempt the optimization of multiple objective functions whereas classical combinatorial optimization only considers the optimization of a single objective function.

The complexity of multi-objective problems lies in the many trade-offs of objectives that can exist. Figure 1.3 shows an example of different trade-offs we could obtain for a pathfinding problem. Let us say we want to travel from city A to city B and want to do so as fast as possible and for as little money as possible. There exists multiple possible routes, the fastest route requires to use expensive transportation while the cheapest route requires to transit in other cities and use slower mean of transportation. Thus, there exists a trade-off between cost and time, the two objectives we want to minimize, and there does not exist one route that is both the cheapest and the fastest one.

Many optimization problems that are NP-complete in the mono-objective case become NP-hard when considered with multiple objectives. Thus, even if most problems actually include

[1] CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=985491

multiple objectives, they are often tackled using mono-objective techniques. Such approach is called *preference-based* multi-objective optimization and requires a pre-processing step where higher-level information are used to determine the importance of each objective. The *weighted-sum* method is the most commonly used preference-based method. It considers a weight for each objective, with a higher weight indicating a higher priority for the objective. Using this set of weights, we can obtain a mono-objective problem where the objective function is the weighted-sum of the multi-objective problem. Another approach called ϵ-*constraint* method requires to select only one objective to actually optimize and then choose a limit for the values of the remaining objectives. We note that both these methods can guarantee to find a solution from the Pareto front. However, they require a difficult step to determine the correct weights or limit to use, as small variations of these parameters can result in very different solutions. These methods are used in practice as they allow the use of existing mono-objective algorithms, but unless the preferences of the decision maker are clear and that the preference-based method is accurate, the resulting solution might not be entirely satisfactory.

In order to be sure that the solution we get is satisfactory, it should be selected from a set of alternatives, ideally the complete Pareto front. Such approach is called *ideal* multi-objective

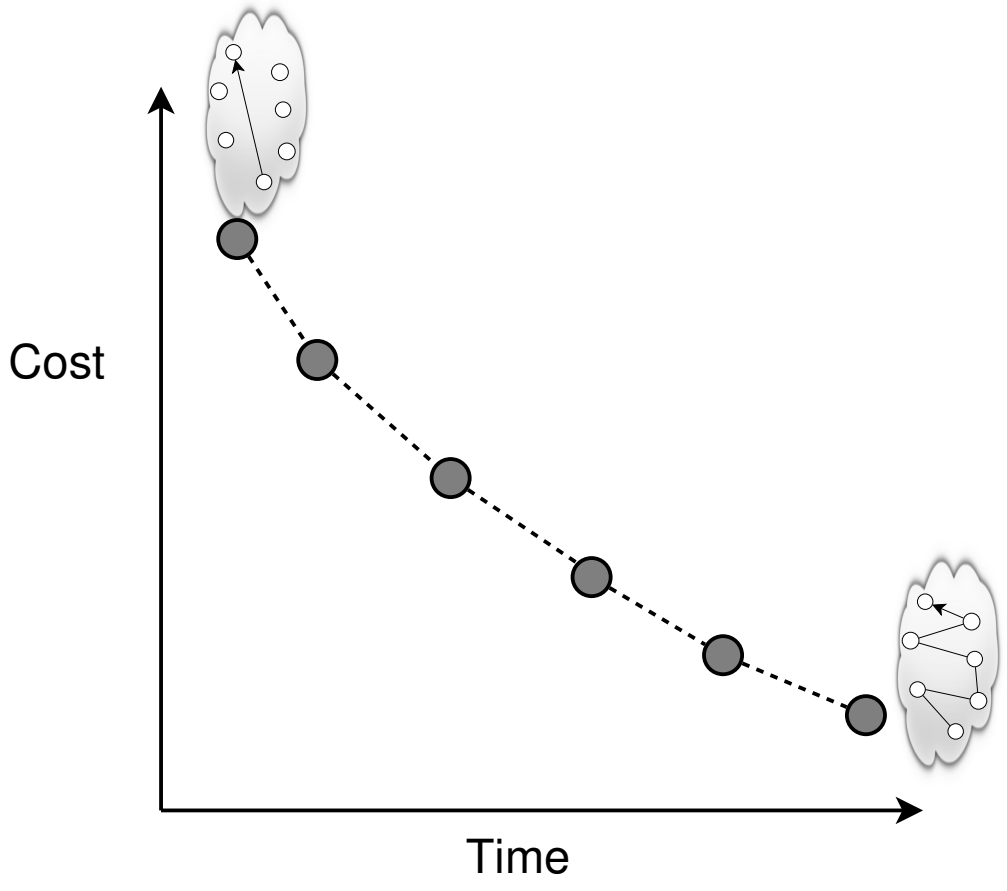
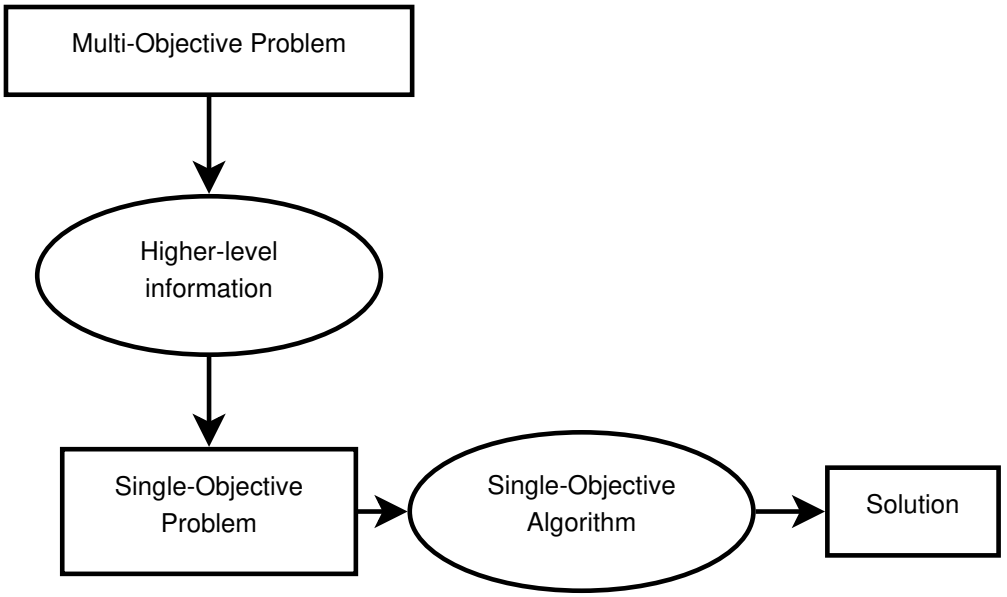
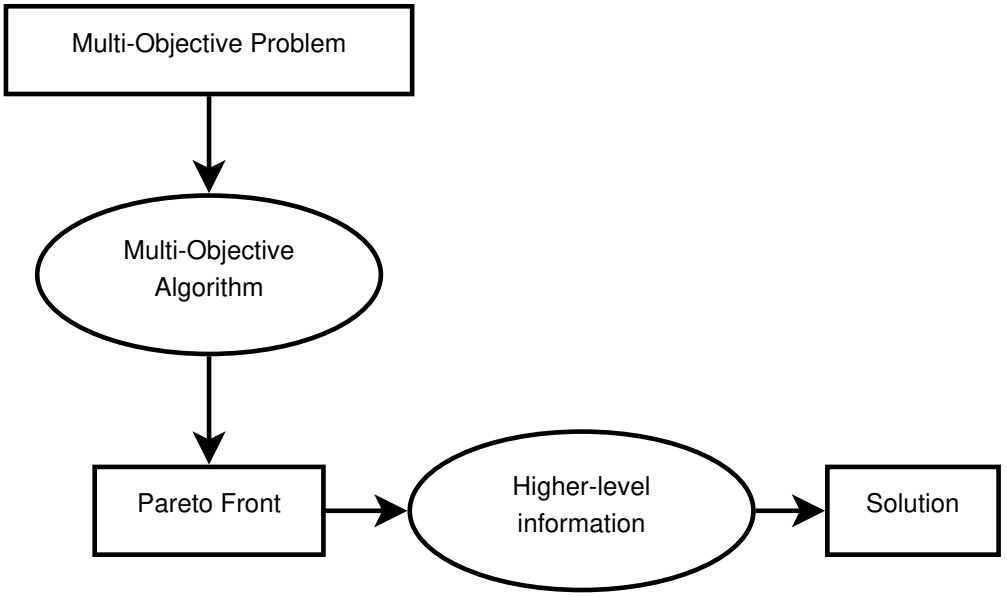


FIGURE 1.3: Example of trade-offs when optimizing the cost and time of a route between two locations.



(a)



(b)

FIGURE 1.4: Illustration of the ideal multi-objective optimization procedure (a) compared to the preference-based multi-objective optimization procedure (b)

optimization [6]. This is the approach we want to study in this thesis. The motivation for this method is that, in order to select the solution of a multi-objective problem, one must have an idea of the different trade-offs of objectives available. Only then can we decide what trade-off to select.

Algorithms for multi-objective problems can be divided into three categories. First, an algorithm that consistently finds the full Pareto front of a given problem is called a *complete* algorithm. An algorithm that finds solutions without being able to guarantee whether they are part of the

Pareto front or not is called an *approximation* algorithm. Finally, an algorithm that can guarantee a subset of the Pareto front is called an *incomplete* algorithm.

The work presented in this thesis contributes to the field of multi-objective optimization in two ways. First, new algorithms are proposed to allow the solving of multi-objective problems in multi-agent systems (discussed in more details in Section 1.1.3). This requires adapting known multi-objective techniques such as local search and dynamic programming to distributed settings, as well as the development of new techniques specific to multi-objective optimization. Secondly, the *ideal* multi-objective optimization approach is studied and new methods are proposed to help the process of selecting a unique solution to a multi-objective problems. This includes considering intermediate approaches where subsets of the Pareto front are extracted in order to simplify the task of the decision maker. The methods proposed in this thesis consider different possible subsets of the Pareto front based on different criteria.

1.1.2 Constraint Programming

Constraint Programming (CP) [7] is an approach to solve many fundamental problems from artificial intelligence. In CP, the modeling part is separated from the solving part. Problems are modeled using constraints, i.e., relations between one or several entities. General search algorithms, usually called *constraint solvers*, are then designed to take advantage of techniques such as constraint propagation or systematic search to find the solution of a given problem.

Originally, CP was limited to the satisfaction of constraints. Meaning that a constraint between some variables is a relation that limit the values simultaneously taken by each variable. *Constraint Satisfaction Problem* [8] (CSP) is the most popular framework to represent satisfaction problems such as the eight queens puzzle, the Sudoku puzzle, or graph-coloring. Many practical problems such as resource allocation or planning can also be represented using CSPs and it has become an important problem in both artificial intelligence and operational research.

Constraint satisfaction however have some limitation. The most important one concerns *over-constrained* problems, i.e., problems where no solution exists that can satisfy all constraints. Such a case is usually the most difficult one to handle by constraint solvers but cannot return any usable solution. Thus, a more general framework called *Weighted Constraint Satisfaction Problem* [9] (WCSP) has been proposed to handle overconstrained CSP. In a WCSP, a weight is associated to each constraint such that if it cannot be satisfied, we have to pay the associated weight. The goal of a WCSP is to find a solution that minimizes the sum of weights. This framework offers a more flexible representation of satisfaction problem, allowing to find a solution for an overconstrained problem but still being able to find a solution fully satisfying the constraints if such a solution exists.

While WCSP provided an important addition to CSP, these two frameworks still use the same type of constraints defining *valid* combinations of values. To allow the representation of problems as general as possible, a new framework designed for optimization was proposed under the name of *Constraint Optimization Problem* [10] (COP). With this framework, constraints are now functions that can yield different costs for each combinations of values of some variables. This framework can still represent the same problems as CSP or WCSP but can also model more complex optimization problems.

The works presented in this thesis make use of constraint frameworks designed specifically to represent multi-objective problems in centralized or distributed settings, namely Multi-Objective Constraint Optimization Problems (MO-COPs) [11], and Distributed Multi-Objective Constraint Optimization Problems (MO-DCOPs) [12]. The algorithms and methods described in this thesis are all designed around these frameworks and contribute with new algorithms to take advantage of the problem decomposition offered by constraint programming.

1.1.3 Distributed Problem Solving

In computer science, a multi-agent system (MAS) [13] is a system composed of a set of agents that interact with each other inside an environment. An agent is characterized by being, at least partially, autonomous. An agent can thus represent a process, a robot, a human, ...

There are two main fields of research on multi-agent systems. First, MAS can be used to simulate complex interactions between autonomous agents in order to analyze the evolution of a system [14]. This has been applied to various fields such as sociology [15], economy [16] or robotics [17]. The second field of research, and the one this thesis belongs to, is distributed problem solving [18]. In distributed problem solving, agents are assumed to collaborate towards achieving a common goal. The problems tackled in distributed problem solving are problems that can also be solved centrally. However, many problems are naturally distributed between agents and using a centralized approach in that case can be inefficient. For example, centralizing information from agents that communicates through wireless communications can produce a tremendous amount of communication overheads. The privacy of agents can also be of concern as some agents might not be willing to share their private information with any other agents. Another advantage of using a distributed approach is with regards to the system's robustness. Centralizing the problem solving incur a single point of failure whereas distributed problem solving can offer methods robust to the loss of some entity of the system.

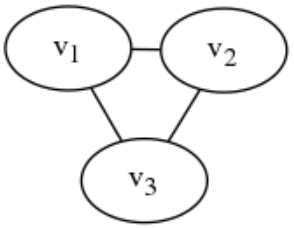
To study distributed problem solving and propose algorithms suited for MAS, specific frameworks have been designed to model decision, satisfaction, and optimization problems among agents. Constraint Satisfaction Problems, where variables share constraints limiting the value they can simultaneously take, have been extended into Distributed Constraint Satisfaction Problems [19] where each variable is controlled by an agent. Partially Observable Markov Decision Processes (POMDPs) [20], used to model decision problems in uncertain environments, have been extended with Decentralized Partially Observable Markov Decision Processes (DEC-POMDPs) [21] to allow the representation of multi-agent coordination problems with uncertain observations and actions. Finally, optimization frameworks like the Constraint Optimization Problem (COP) [22] have been extended for MAS with the Distributed Constraint Optimization Problem (DCOP) [23, 24].

This thesis contributes to the field of distributed problem solving by proposing new distributed algorithms to allow agents to collaborate and achieve multiple objectives.

Chapter 2

Preliminaries

In this chapter, we introduce the preliminary notions required for understanding the contribution of this thesis. Section 2.1 presents the basic model used to represent the problems tackled in our work, namely the model for a Constraint Optimization problem (COP). Section 2.2 presents the Multi-Objective Constraint Optimization Problem (MO-COP), the multi-objective extension of a COP. Section 2.3 discusses the extra considerations required in multi-agent systems and presents the distributed version of an MO-COP, namely, the model for a Multi-Objective Distributed Constraint Optimization Problem (MO-DCOP). Section 2.4 presents the model for a dynamic MO-COP. Section 2.5 presents some examples of applications for the models presented previously. Finally, Section 2.6 presents a summary of this thesis' contribution.



v_1	v_2	$cost$	v_2	v_3	$cost$	v_1	v_3	$cost$
a	a	5	a	a	0	a	a	1
a	b	7	a	b	2	a	b	1
b	a	10	b	a	0	b	a	0
b	b	12	b	b	2	b	b	3

FIGURE 2.1: Example of Constraint Optimization Problem.

2.1 Constraint Optimization Problems

Definition 2.1 (Constraint Optimization Problem)**.** A *Constraint Optimization Problem* (COP) [25, 26] is the problem of finding an assignment of values to some variables so that the sum of the resulting costs is minimized.

A COP is defined as a tuple $(V, \mathcal{D}, \mathcal{C}, F)$ such that:

- $V = \{v_1, \ldots, v_n\}$ is a set of variables;
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of domains;
- $\mathcal{C} = \{C_1, \ldots, C_c\}$ is a set of constraint relations;
- $F = \{f_1, \ldots, f_c\}$ is a set of cost functions.

A variable $v_i \in V$ takes its value from a finite, discrete domain $D_i \in \mathcal{D}$. A constraint relation $C_i \in \mathcal{C}, C_i \subseteq V$ is a set of variables, indicating that the variables in C_i share a constraint relation.

For each constraint relation $C_i \in \mathcal{C}$, there is a corresponding cost function f_i defining a cost for each combination of values of the variables in C_i, i.e., a cost function $f_i : \times_{\forall v_j \in C_i} D_j \rightarrow \mathbb{R}_{\geq 0}, f_i \in F$.

In a COP, an assignment A is a set of pairs of variable/value, written $\{(v_i, d_i), (v_j, d_j), \ldots\}$, indicating that variable $v_i \in V$ takes value $d_i \in D_i$. If a variable v_i is part of assignment A, we can say that v_i is in A, written $v_i \in A$.

The cost of an assignment A can be computed as follows:

$$R(A) = \sum_{C_i \in \mathcal{C} \ s.t. \ C_i \subseteq A} f_i(A)$$

If an assignment A contains all variables of the problem, i.e., $\forall v_i \in V, v_i \in A$, then A is said to be a *complete assignment*.

An *optimal assignment* A^* is given as $\arg\min_{A=V} R(A)$, meaning there does not exist another complete assignment with a better cost. The goal of a COP is to find an *optimal assignment*.

Remark 2.2. COP is NP-hard.

Remark 2.3 (Relation with CSPs). Constraint Satisfaction Problems (CSP) can be represented as COPs where cost functions yield a cost of 0 for valid assignments and a cost of 1 for invalid assignments.

Example 2.1 (COP). *Figure 2.1 shows a COP* $(V, \mathcal{D}, \mathcal{C}, F)$ *as follows:*

- $V = \{v_1, v_2, v_3\}$.
- $\mathcal{D} = \{D_1, D_2, D_3\}$ *where* $D_1 = D_2 = D_3 = \{a, b\}$;
- $\mathcal{C} = \{C_1, C_2, C_3\}$ *where* $C_1 = \{v_1, v_2\}$, $C_2 = \{v_1, v_3\}$, $C_3 = \{v_2, v_3\}$;
- $F = \{f_1, f_2, f_3\}$ *where each function is represented as a table in Figure 2.1.*

All possible assignments and their resulting costs are shown in Table 2.1. The optimal solution of this problem is $\{(v_1, a), (v_2, a), (v_3, a)\}$ *with an optimal cost of 6.*

This cost is computed by summing the cost of each cost function for the assignment:

- *For* $C_1 = \{v_1, v_2\}$, $f_1(\{(v_1, a), (v_2, a)\}) = 5$.
- *For* $C_2 = \{v_1, v_3\}$, $f_2(\{(v_1, a), (v_3, a)\}) = 0$.
- *For* $C_3 = \{v_2, v_3\}$, $f_3(\{(v_2, a), (v_3, a)\}) = 1$.
- $R(\{(v_1, a), (v_2, a), (v_3, a)\}) = 6$.

v_1	v_2	v_3	Cost
a	a	a	6
a	a	b	8
a	b	a	8
a	b	b	10
b	a	a	10
b	a	b	15
b	b	a	12
b	b	b	17

TABLE 2.1: Possible solutions of the COP of Figure 2.1.

2.2 Multi-Objective Constraint Optimization Problems

Definition 2.4 (Multi-Objective Constraint Optimization Problem)**.** A *Multi-Objective Constraint Optimization Problem* (MO-COP) [27–29] is the problem of finding an assignment of values to some variables so that the sum of the resulting costs is minimized. It is the extension of a (mono-objective) Constraint Optimization Problem (COP) where, instead of having a single cost function per constraint relation, multiple functions corresponding to multiple objectives are used.

An MO-COP is defined as a tuple $(V, \mathcal{D}, \mathcal{C}, \mathcal{F})$ such that:

- $V = \{v_1, \ldots, v_n\}$ is a set of variables;
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of domains;
- $\mathcal{C} = \{C_1, \ldots, C_c\}$ is a set of constraint relations;
- $\mathcal{F} = \{F_1, \ldots, F_c\}$ is a set of sets of cost functions.

A variable $v_i \in V$ takes its value from a finite, discrete domain $D_i \in \mathcal{D}$. A constraint relation $C_i \in \mathcal{C}, C_i \subseteq V$ is a set of variables, indicating that the variables in C_i share a constraint relation.

For each constraint relation $C_i \in \mathcal{C}$, there is a corresponding set of cost functions $F_i = \{f_1, \ldots, f_m\}$ providing a cost function for each objective of the problem. Then, each cost function $f_l \in F_i$ defines a cost for each combination of values of the variables in C_i, i.e., a cost function $f_l : \times_{\forall v_j \in C_i} D_j \rightarrow \mathbb{R}_{\geq 0}, f_l \in F_i$.

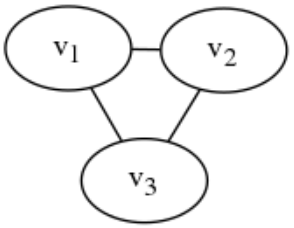
Similarly to COPs, in a MO-COP, an assignment A is a set of pairs of variable/value, written $\{(v_i, d_i), (v_j, d_j), \ldots\}$, indicating that variable $v_i \in V$ takes value $d_i \in D_i$. If a variable v_i is part of assignment A, we can say that v_i is in A, written $v_i \in A$.

The cost for objective l of an assignment A can be computed as follows:

$$R^l(A) = \sum_{C_i \in \mathcal{C} \text{ s.t. } C_i \subseteq A} f_l(A), f_l \in F_i$$

Then, the sum of the values of all cost functions for m objectives is defined by a cost vector, denoted $R(A) = (R^1(A), R^2(A), \ldots, R^m(A))$.

To find an assignment that minimizes all m objective functions simultaneously is ideal. However, in general, since trade-offs exist among objectives, there does not exist such an ideal assignment. Therefore, the optimal solution of an MO-COP is characterized by using the concept



v_1	v_2	$cost$	v_2	v_3	$cost$	v_1	v_3	$cost$
a	a	(5,2)	a	a	(0,1)	a	a	(1,0)
a	b	(7,1)	a	b	(2,1)	a	b	(1,0)
b	a	(10,3)	b	a	(0,2)	b	a	(0,1)
b	b	(12,0)	b	b	(2,0)	b	b	(3,2)

FIGURE 2.2: Example of Multi-Objective Constraint Optimization Problem.

of *Pareto optimality*. An assignment is a *Pareto optimal solution* if there does not exist another assignment that weakly improves all of the objectives.

Solving an MO-COP is to find the *Pareto front* which is a set of cost vectors obtained by all Pareto optimal solutions.

Definition 2.5 (Pareto Dominance)**.** For an MO-COP and two cost vectors $R(A)$ and $R(A')$, we say that $R(A)$ dominates $R(A')$, denoted by $R(A) \prec R(A')$, iff $R(A)$ is partially less than $R(A')$, i.e., it holds (i) $R^h(A) \leq R^h(A')$ for all objectives h, and (ii) there exists at least one objective h', such that $R^{h'}(A) < R^{h'}(A')$.

Example 2.2 (Pareto Dominance)**.** *Let us consider 3 vectors with 2 objectives:*

- $R_1 = (6, 3)$.
- $R_2 = (10, 1)$.
- $R_3 = (17, 2)$.

We can notice the following relationships:

- $R_2 \prec R_3$*, i.e.,* $R_2 = (10, 1)$ *dominates* $R_3 = (17, 2)$.
- $R_1 \not\prec R_2$ *and* $R_1 \not\succ R_2$.

Definition 2.6 (Pareto optimal solution)**.** For an MO-COP, an assignment A is said to be a Pareto optimal solution, if and only if there does not exist another assignment A', such that $R(A') \prec R(A)$.

Definition 2.7 (Pareto Front)**.** For an MO-COP, a set of cost vectors obtained by the Pareto optimal solutions is called the Pareto front.

Example 2.3 (MO-COP)**.** *Table 2.4 shows a bi-objective COP, which is an extension of the COP in Figure 2.1. Each variable takes its value from a discrete domain* $\{a, b\}$*. The Pareto optimal solutions of this problem are* $\{\{(v_1, a), (v_2, a), (v_3, a)\}, \{(v_1, a), (v_2, b), (v_3, b)\}\}$*, and the Pareto front is* $\{(6, 3), (10, 1)\}$.

Example 2.4 (MO-COP). *Figure 2.2 shows an MO-COP* $(V, \mathcal{D}, \mathcal{C}, \mathcal{F})$ *as follows:*

- $V = \{v_1, v_2, v_3\}$.
- $\mathcal{D} = \{D_1, D_2, D_3\}$ *where* $D_1 = D_2 = D_3 = \{a, b\}$;
- $\mathcal{C} = \{C_1, C_2, C_3\}$ *where* $C_1 = \{v_1, v_2\}$, $C_2 = \{v_1, v_3\}$, $C_3 = \{v_2, v_3\}$;
- $\mathcal{F} = \{F_1, F_2, F_3\}$ *where each set of functions is represented as a table in Figure 2.2.*

All possible assignments and their resulting cost vectors are shown in Table 2.2. The Pareto solutions of this MO-COP are:

- $S_1 = \{(v_1, a), (v_2, a), (v_3, a)\}$ *with cost* $R(S_1) = (6, 3)$.
- $S_2 = \{(v_1, a), (v_2, b), (v_3, b)\}$ *with cost* $R(S_2) = (10, 1)$.

v_1	v_2	v_3	Cost
a	a	a	(6,3)
a	a	b	(8,3)
a	b	a	(8,3)
a	b	b	(10,1)
b	a	a	(10,5)
b	a	b	(15,6)
b	b	a	(12,3)
b	b	b	(17,2)

TABLE 2.2: Possible solutions of the MO-COP of Figure 2.2.

2.3 Distributed Optimization

In this section, we present the distributed extensions of the Constraint Optimization Problem presented in Section 2.1 and of the Multi-Objective Constraint Optimization Problem presented in Section 2.2.

We will first present in Section 2.3.1 the assumptions and considerations we make for distributed systems. We then present the frameworks for the Distributed Constraint Optimization Problem in Section 2.3 and for the Multi-Objective Distributed Constraint Optimization Problem in Section 2.3.3. We also present the concept of pseudo-tree in Section 2.3.4 as it is a structure commonly used by distributed algorithms.

2.3.1 Assumptions and Considerations

Classical optimization is assumed to be performed in a centralized fashion, where a single agent knows the whole problem and performs all the computation. However, many problems are naturally distributed with each agent being in relation with other agents to form global constraint problems.

This introduces many new considerations about communication cost, privacy or the distribution of the computation.

2.3.1.1 Ownership and Control

Compared to the Constraint Optimization Problem (assumed to be centralized), a DCOP considers that each variable of the problem is managed by an agent. In practice, each agent can be responsible of multiple variables but to simplify notations, we assume that an agent x_i owns a single *meta* variable v_i that actually represents the values of multiple local variables by taking values from the set of all their possible combinations.

2.3.1.2 Cooperation

In this thesis, we only consider *cooperative* agents that all work toward the same global goal. This is what is represented with the framework we use called Distributed Constraint Optimization Problem and that we present in Section 2.3.

2.3.1.3 Privacy

In a distributed problem, each agent knows a different part of the subproblem and how these information are exchanged is a critical part of distributed algorithms. Throughout this thesis, we will assume that an agent x_i is only aware of the following:

- Its variable v_i and the corresponding domain D_i.
- The set of constraints $\mathcal{C}^i$ including variable v_i and their corresponding set of cost functions F^i.
- The set of variables sharing a constraint with v_i and their corresponding domains.

2.3.2 Distributed Constraint Optimization Problems

In this subsection, we describe the formalism of Distributed Constraint Optimization Problems (DCOPs).

Definition 2.8 (Distributed Constraint Optimization Problem)**.** A *Distributed Constraint Optimization Problem* (DCOP) [25, 26] is the problem of finding an assignment of values to some variables so that the sum of the resulting costs is minimized.

A DCOP is defined as a tuple $(X, V, \mathcal{D}, \mathcal{C}, F)$ such that:

- $X = \{x_1, \ldots, x_n\}$ is a set of agents;
- $V = \{v_1, \ldots, v_n\}$ is a set of variables;
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of domains;
- $\mathcal{C} = \{C_1, \ldots, C_c\}$ is a set of constraint relations;
- $F = \{f_1, \ldots, f_c\}$ is a set of cost functions.

An agent $x_i \in X$ is responsible of variable $v_i \in V$ and chooses its value from a finite, discrete domain $D_i \in \mathcal{D}$.

A constraint relation $C_i \in \mathcal{C}, C_i \subseteq V$ is a set of variables, indicating that the variables in C_i share a constraint relation.

For each constraint relation $C_i \in \mathcal{C}$, there is a corresponding cost function f_i defining a cost for each combination of values of the variables in C_i, i.e., a cost function $f_i : \times_{\forall v_j \in C_i} D_j \rightarrow \mathbb{R}_{\geq 0}, f_i \in F$.

In a DCOP, an assignment A is a set of pairs of variable/value, written $\{(v_i, d_i), (v_j, d_j), \ldots\}$, indicating that variable $v_i \in V$ takes value $d_i \in D_i$. If a variable v_i is part of assignment A, we can say that v_i is in A, written $v_i \in A$.

The cost of an assignment A can be computed as follows:

$$R(A) = \sum_{C_i \in \mathcal{C} \ s.t. \ C_i \subseteq A} f_i(A)$$

If an assignment A contains all variables of the problem, i.e., $\forall v_i \in V, v_i \in A$, then A is said to be a *complete assignment*.

An *optimal assignment* A^* is given as $\arg\min_{A=V} R(A)$, meaning there does not exist another complete assignment with a better cost. The goal of a DCOP is to find *one optimal assignment*.

2.3.3 Multi-Objective Distributed Constraint Optimization Problems

Definition 2.9 (Multi-Objective Distributed Constraint Optimization Problem)**.** A *Multi-Objective Distributed Constraint Optimization Problem* (MO-DCOP) [12, 30, 31] is the problem of finding an assignment of values to some variables so that the sum of the resulting costs is minimized. It is the extension of a (mono-objective) Constraint Optimization Problem (COP) where, instead of having a single cost function per constraint relation, multiple functions corresponding to multiple objectives are used.

An MO-DCOP is defined as a tuple $(X, V, \mathcal{D}, \mathcal{C}, \mathcal{F})$ such that:

- $X = \{x_1, \ldots, x_n\}$ is a set of agents;
- $V = \{v_1, \ldots, v_n\}$ is a set of variables;
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of domains;
- $\mathcal{C} = \{C_1, \ldots, C_c\}$ is a set of constraint relations;
- $\mathcal{F} = \{F_1, \ldots, F_c\}$ is a set of sets of cost functions.

An agent $x_i \in X$ is responsible of variable $v_i \in V$ and chooses its value from a finite, discrete domain $D_i \in \mathcal{D}$.

A constraint relation $C_i \in \mathcal{C}, C_i \subseteq V$ is a set of variables, indicating that the variables in C_i share a constraint relation.

For each constraint relation $C_i \in \mathcal{C}$, there is a corresponding set of cost functions $F_i = \{f_1, \ldots, f_m\}$ providing a cost function for each objective of the problem. Then, each cost

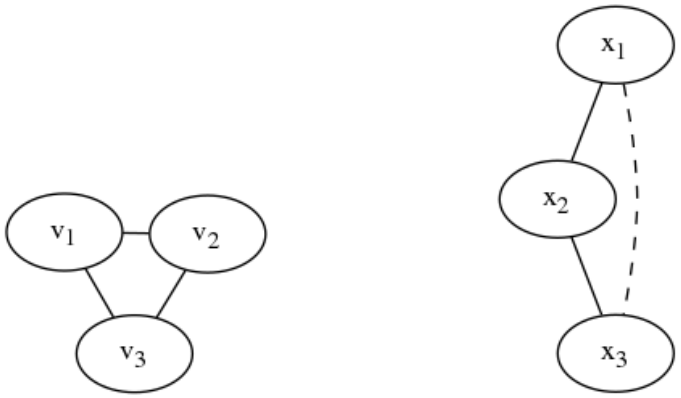


FIGURE 2.3: Example of constraint graph (left) and the corresponding pseudo-tree (right)

function $f_l \in F_i$ defines a cost for each combination of values of the variables in C_i, i.e., a cost function $f_l : \times_{\forall v_j \in C_i} D_j \rightarrow \mathbb{R}_{\geq 0}, f_l \in F_i$.

Similarly to DCOPs, in a MO-DCOP, an assignment A is a set of pairs of variable/value, written $\{(v_i, d_i), (v_j, d_j), \ldots\}$, indicating that variable $v_i \in V$ takes value $d_i \in D_i$. If a variable v_i is part of assignment A, we can say that v_i is in A, written $v_i \in A$.

The cost for objective l of an assignment A can be computed as follows:

$$R^l(A) = \sum_{C_i \in \mathcal{C} \text{ s.t. } C_i \subseteq A} f_l(A), f_l \in F_i$$

Then, the sum of the values of all cost functions for m objectives is defined by a cost vector, denoted $R(A) = (R^1(A), R^2(A), \ldots, R^m(A))$.

Similarly to MO-COPs, we define the goal of an MO-DCOP as finding its Pareto front (Definition 2.7).

2.3.4 Pseudo-Tree

In many DCOP algorithms, agent are organized using a *pseudo-tree* [26] where there exists a unique root node and each non-root node has a parent node. In addition, it is required that all variables sharing a constraint must be part of a same path between the root and a leaf. Such structure can be obtained using a depth-first traversal of the constraint graph.

After such structure is generated, each agent x_i is aware of its parent P_i, its children CH_i, and its pseudo-parents PP_i. An agent x_j is a pseudo-parent of x_i if and only if it is an ancestor of P_i in the pseudo-tree and a neighbor of x_i in the constraint graph.

An important concept of pseudo-trees for the algorithms discussed in this thesis is the *separator* of an agent.

Definition 2.10 (Separator)**.** In a pseudo-tree, the separator Sep_i of a node x_i is the set of all ancestors of x_i which are pseudo-parents of x_i or its descendants.

$$\text{Sep}_i = \text{Ancestors}_i \cap (\text{PP}_i \cup (\bigcup_{x_j \in \text{Descendants}_i} \text{PP}_j)$$

Example 2.5 (Pseudo-tree)**.** *Figure 2.3 shows a constraint graph and the corresponding pseudo-tree. Lines represent parent/child relations and dotted lines represent pseudo-parent relations. With x_1 the root of the tree, the separators of this example are $Sep_1 = \emptyset, Sep_2 = \{x_1\}$, and $Sep_3 = \{x_1, x_2\}$.*

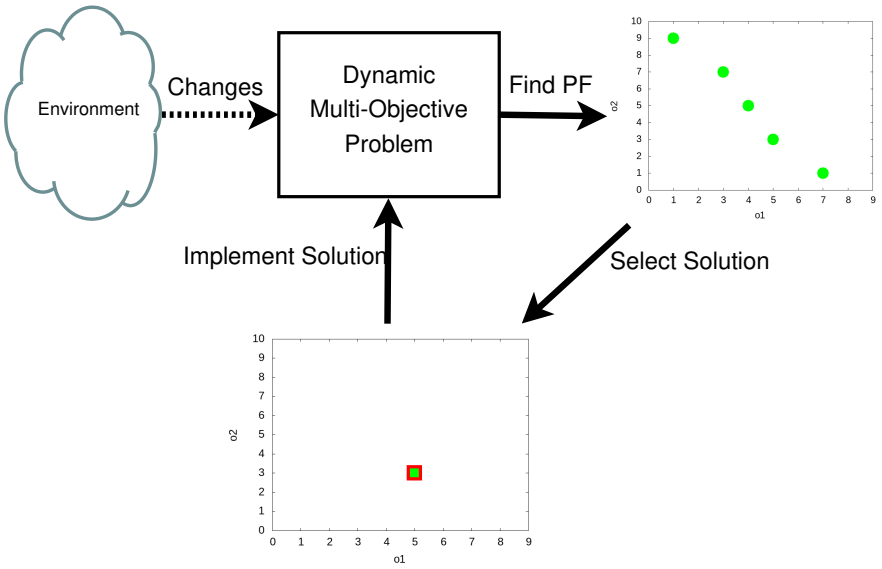


FIGURE 2.4: Dynamic cycle of (i) having a problem, (ii) finding its Pareto front, and (iii) selecting a solution to implement. This cycle must be repeated each time the problem changes.

2.4 Dynamic Multi-Objective Constraint Optimization Problems

We now present the framework for a *Dynamic Multi-Objective Constraint Optimization Problem* (DMO-COP).

Definition 2.11 (Dynamic MO-COP)**.** A DMO-COP is defined by a sequence of MO-COPs as follows

where each index i ($0 \leq i \leq k$) represents a time step. Solving a DMO-COP is finding the following sequence of Pareto front, denoted $\mathbb{PF}$, where each PF_i ($0 \leq i \leq k$) represents the Pareto front of MO-COP$_i$.

$$\mathbb{PF} = \langle \mathrm{PF}_0, \mathrm{PF}_1, ..., \mathrm{PF}_\mathrm{k} \rangle.$$

Such framework is used to model problems that change over time, requiring new solutions after each change. Using this framework, after changes occur, a completely new problem is assumed.

Two cases are possible when considering dynamic problems. First, a *reactive* case where future changes are unknown, meaning that a problem $MO\text{-}COP_i$ is unknown until the problem $MO\text{-}COP_{i-1}$ is solved (except for the first problem in the sequence corresponding to $i = 0$). The other case is a *proactive* case where the future changes are known in advance, meaning that the whole sequence is given.

2.5 Applications

Many problems can be modeled as Constraint Optimization Problems. Classical problems include the vertex cover problem [32], the knapsack problem [3, 4], vehicle routing [33] or the MaxSAT problem [34].

Distributed Constraint Optimization Problems have been applied to a wide variety of multi-agent coordination problems. The most notable applications include sensor networks [35, 36], distributed meeting scheduling [37–39], coalition structure generation [40], or the synchronization of traffic lights [41].

While few works have tackled multi-objective problems in distributed environments, all mono-objective applications can potentially be considered. The one existing application is toward the optimization of a water resource management system [42].

2.5.1 Timetabling

In this section, we describe the model of Curriculum-Based Course Timetabling [43] (CB-CTT) as it was defined for the ITC-2007 competition [44].

CB-CTT models the course timetabling problem that arises in many universities. In this model, we have a set of curricula that predefines the sets of courses a student can follow. Then, each course is made of several lectures that will take place every week. One of the simplification made compared to other models of timetabling is the omission of student subsectioning, where each student has to be assigned to individual subsections of a course. Even with this simplification, CB-CTT is an NP-hard problem.

In this model, we are given sets of:

- *time periods*: the time horizon is divided into days and time slots per day. A time period is a pair (day, time slot).
- *courses*: each course consists of a given number of lectures, is taught by a teacher and is attended by a given number of students.
- *curricula*: a curriculum is a set of courses.
- *rooms*: each room has a maximum capacity.

CB-CTT then consists in finding an assignment of course lectures to rooms while satisfying a set of hard-constraints:

- H_1. **Lectures**: all lectures of each course must be scheduled and they must be assigned to distinct time slots.
- H_2. **Conflicts**: lectures of courses in the same curriculum or taught by the same teacher must be all scheduled in different time slots.
- H_3. **Room Occupancy**: two lectures can not take place in the same room in the same time slot.
- H_4. **Availability**: if the teacher of the course is not available to teach that course at a given time slot, then no lecture of the course can be scheduled at that time slot.

Definition 2.12 (Valid timetable)**.** A timetable T is valid if it satisfies all hard-constraints.

To be able to compare different valid timetables, other constraints qualified as soft-constraints, are used as quality measures:

- S_1. **Room Capacity**: For each lecture, the number of students that attend the course must be less than or equal the number of seats of all the rooms that host its lectures. The penalties, reflecting the number of students above the capacity, are imposed on each violation.
- S_2. **Minimum Working Days**: The lectures of each course must be spread into a given minimum number of days. The penalties, reflecting the number of days below the minimum, are imposed on each violation.
- S_3. **Isolated Lectures**: Lectures belonging to a curriculum should be adjacent to each other in consecutive timeslots. For a given curriculum we account for a violation every time there is one lecture not adjacent to any other lecture within the same day. Each isolated lecture in a curriculum counts as one violation.
- S_4. **Windows**: Lectures belonging to a curriculum should not have time windows (periods without teaching) between them. For a given curriculum we account for a violation every time there is one window between two lectures within the same day. The penalties, reflecting the length in periods of time window, are imposed on each violation.
- S_5. **Room Stability**: All lectures of a course should be given in the same room. The penalties, reflecting the number of distinct rooms but the first, are imposed on each violation.
- S_6. **Student MinMax Load**: For each curriculum the number of daily lectures should be within a given range. The penalties, reflecting the number of lectures below the minimum or above the maximum, are imposed on each violation.

TABLE 2.3: Formulations of CB-CTT

Constraint	UD_1 [45]	UD_2 [46]	UD_3 [47]	UD_4 [47]	UD_5 [47]
H_1 : Lectures	H	H	H	H	H
H_2 : Conflicts	H	H	H	H	H
H_3 : Room Occupancy	H	H	H	H	H
H_4 : Availability	H	H	H	H	H
S_1 : Room Capacity	1	1	1	1	1
S_2 : Minimum Working Days	5	5	0	1	5
S_3 : Isolated Lectures	1	2	0	0	1
S_4 : Windows	0	0	4	1	2
S_5 : Room Stability	0	1	0	0	0
S_6 : Student MinMax Load	0	0	2	1	2
S_7 : Travel Distance	0	0	0	0	2
S_8 : Room Suitability	0	0	3	H	0
S_9 : Double Lectures	0	0	0	1	0

- S_7. **Travel Distance**: Students should have the time to move from one building to another one between two lectures. For a given curriculum we account for a penalty every time there is an instantaneous move: two lectures in rooms located in different building in two adjacent periods within the same day. Each instantaneous move in a curriculum counts as one penalty.

- S_8. **Room Suitability**: Some rooms may not be suitable for a given course because of the absence of necessary equipment. Each lecture of a course in an unsuitable room counts as one penalty.

- S_9. **Double Lectures**: Some courses require that lectures in the same day are grouped together (double lectures). For a course that requires grouped lectures, every time there is more than one lecture in one day, a lecture non-grouped to another is not allowed. Two lectures are grouped if they are adjacent and in the same room. Each non-grouped lecture counts as one penalty.

Using those soft-constraints, we can represent the quality of a timetable T as a vector of penalties $V(T) = (v_1, v_2, \ldots, v_9)$ where v_i is the number of penalties for constraint S_i ($1 \leq i \leq 9$). Then, if we know the preferences of the decision makers, we can associate a weight to each soft-constraint such that the lower the weight, the lower the importance of the constraint. A weight of 0 corresponds to completely ignoring the constraint.

Various formulations of the problem uses different weights, as seen in Table 2.3. In the table, lines represent constraints and columns represent the various formulations (UD_1 to UD_5). A "H" in a cell indicates that the formulation uses the constraint as a hard-constraint. A number in a cell indicates the weight associated with the soft-constraint. UD_1 is the first set of weights proposed for the CB-CTT problem [45] and only considers three soft-constraints. UD_2 is the

set of weights proposed for the ITC-2007 competition [46], where they extend UD_1 with one additional soft-constraint. UD_3, UD_4, and UD_5 [47], were proposed in a latter work in order to offer variations of the original problem by considering 4 new soft-constraints. Those five formulations are the most commonly studied and are the ones we will use throughout this paper. In real applications however, those formulations might not be directly used since different universities might want to consider different constraints and weights.

Using a formulation UD_x with the associated set of weights W_x, we can transform a vector a penalties $V(T)$ into a weighted vector $V_x(T)$.

Definition 2.13 (Weighted-Vector)**.** Given a timetable T and a set of weights W_x corresponding to formulation UD_x, we have the corresponding weighted-vector $V_x(T) = (v_1 \times w_1, v_2 \times w_2, \ldots, v_9 \times w_9)$ with $v_i \in V(T)$.

The most common approach to the CB-CTT problem aims at finding a solution such that the sum of values in its corresponding weighted-vector is minimized. We call such solution a sum-optimal solution and it is the typical result to scalarization methods.

Definition 2.14 (sum-Optimal solution)**.** Given a valid timetable T and a formulation UD_x, we say T is sum-optimal if it minimizes

$$\sum_{v_i \in V_x(T)} v_i$$

CB-CTT can be represented as a Multi-Objective Constraint Optimization Problems where:

- Each lecture of a course is a variable.
- The domain of a variable is the set of all combinations of room $\times$ period.

Constraints can then naturally expressed between variables and we refer to the paper by Banbara et al. [48] for a detailed COP formalization.

2.5.2 Multi-Objective Graph Coloring

Graph-coloring [49] is a classical problem in optimization where, given a graph and a limited number of colors, one must assign a color to each node of the graph such that no neighboring nodes share a same color.

Existing works on MO-DCOPs experiments with a multi-objective version of the graph-coloring problem [12, 30]. While this is an abstract problem, it is used in real applications such as scheduling problems [50] and wireless sensor networks [36]. In this problem, each agent x_i

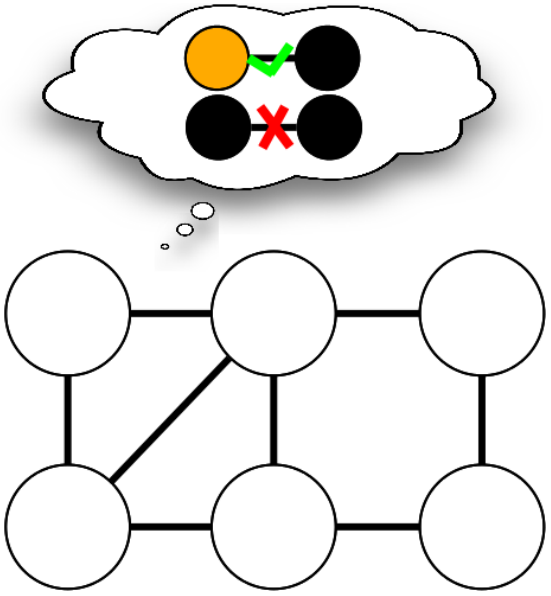


FIGURE 2.5: Illustration of the graph-coloring problem.

owns a variable v_i taking its value from the domain $D_i = \{1, 2, 3\}$. Three cost functions corresponding to three objectives are defined:

$$f^1(v_i, v_j) = \begin{cases} 0 & v_i \neq v_j \\ 1 & v_i = v_j \end{cases}, f^2(v_i, v_j) = \begin{cases} 0 & \text{if } |v_i - v_j| = 1 \\ 1 & \text{otherwise} \end{cases},$$

$$f^3(v_i, v_j) = \begin{cases} 0 & \text{if } i < j \text{ and } v_i < v_j \\ 1 & \text{otherwise} \end{cases}$$

Chromatic Difference : this objective function represents the common graph coloring conflict function.

Chromatic Ordering : this objective function imposes an ordering among the colors: Red = 1, Green = 2, and Blue = 3. Specifically, given two variables v_i and v_j where $i < j$, the variable with the higher index should have a higher ranked color.

Chromatic Distance : this objective is similar to the chromatic ordering. However, it considers the distance between the colors of different variables. In more detail, given two variables v_i and v_j, the distance of the colors should equal one.

2.6 Contribution

We now present the different contributions of this thesis.

Most problems include multiple conflicting objectives that should be optimized simultaneously. Finding the Pareto front of multi-objective problem is a challenging problem but is a necessary step in an ideal multi-objective optimization procedure.

The first part of our contribution proposes *distributed* algorithms to find the Pareto front of multi-objective problems in a multi-agent system. At the time of writing this thesis, few algorithms have been proposed to find the Pareto front of Multi-Objective Distributed Constraint Optimization Problems (MO-DCOP). A large part of this thesis thus contributes to this area by proposing three distributed algorithms.

- The Multi-Objective Distributed Pseudo-tree Optimization (MO-DPOP) is a *complete* algorithm for MO-DCOP making use of dynamic programming techniques.
- The Bounded Multi-Objective Distributed Pseudo-tree Optimization (MO-DPOP(B)) is an extension of MO-DPOP where a function is used to bound the messages communicated during the algorithm, offering an *approximation* algorithm for MO-DCOP. For specific bounding functions, MO-DPOP(B) can become an *incomplete* algorithm.
- The Distributed Pareto Local Search (DPLS) is an *approximation* algorithm for MO-DCOP making use of local search techniques.

These algorithms can be used to find a set of trade-off solutions in an MO-DCOP. If we follow the *ideal* multi-objective optimization procedure, a decision still has to be made to select a single solution from the set of trade-offs found by the MO-DCOP algorithms. Thus, the second part of our contribution relates to multi-objective selection methods used to select a single solution in a multi-objective problem. To select or help select a solution from a Pareto front, we propose three methods.

- A sum-based method which uses the weighted-sum, the most popular technique for preference-based multi-objective optimization. Whereas this method is usually used to transform a multi-objective problem into a single-objective problem, we here propose to find *all* solutions that optimize the weighted-sum, providing a subset of the Pareto front.
- A resilience-based method which isolates solutions that satisfy some given resilience properties.
- A transition-based method which considers the cost of implementing a solution and select a solution such that its implementation cost respects a given limit.

Most of the contribution presented in this thesis have been published in conference proceedings or in journals.

- The work on DPLS titled "Distributed Pareto Local Search for Multi-Objective DCOPs" has been accepted for publication in the Journal of the Institute of Electronics, Information and Communication: Special issue on Frontiers in Agent-based Technology [51].
- The work on our sum-based solution selection method titled "$\sum_x$-Optimal Solutions in Highly Symmetric Multi-Objective Timetabling Problems" appeared in the proceeding of the 11th International Conference on the Practice and Theory of Automated Timetabling [52].
- The work on our resilience-based solution selection method titled "Finding Resilient Solutions for Dynamic Multi-Objective Constraint Optimization Problems" appeared in the proceedings of the 7th International Conference on Agents and Artificial Intelligence [53].
- The work on our transition-based solution selection method titled "Limiting Perturbations in Dynamic DCOP" was presented at the 30th Annual Conference of the Japanese Society for Artificial Intelligence [54].

Chapter 3

Algorithms for Multi-Objective Distributed Constraint Optimization Problems

In this chapter, we present new algorithms for finding the Pareto front of a Multi-Objective Distributed Constraint Optimization Problem (MO-DCOP). This corresponds to the first step of the solving of multi-objective problems where we take an abstract multi-objective problem and output its Pareto front.

We propose here two algorithms dedicated to distributed problems. The first algorithm presented in section 3.1 is the Multi-Objective Distributed Pseudo-tree Optimization (MO-DPOP), a complete algorithm that guarantees to find the full Pareto front of an MO-DCOP. The second algorithm presented in section 3.2 is the Distributed Pareto Local Search (DPLS), an algorithm using local search to approximate the Pareto front of an MO-DCOP.

3.1 Multi-Objective Distributed Pseudo-tree Optimization Procedure

In this section, we develop a novel complete algorithm for MO-DCOPs called *Multi-Objective Distributed Pseudo-tree Optimization Procedure* (MO-DPOP). This algorithm extends DPOP, the representative inference algorithm for DCOPs. MO-DCOP algorithms can be divided into two groups, i.e., search and inference algorithms. MO-ADOPT, the state-of-the-art complete algorithm, is a search algorithm while our proposed algorithm is an inference-based algorithm. In mono-objective DCOPs, it is well-known that the size of the messages of inference algorithms such as DPOP can be exponential in the number of variables, which is a disadvantage compared to search algorithms. However, in MO-DCOPs, this disadvantage can be ignored since it also becomes exponential for search algorithms, i.e., there is no advantage to use a search algorithm for MO-DCOPs, as we will show in our experiments. Furthermore, we provide an incomplete method to improve the performances of MO-DPOP by bounding the size of the messages exchanged between the agents. Compared to the state of the art approximation algorithm for MO-DCOPs (B-MOMS [12]), our incomplete algorithm can find a subset of the Pareto front, i.e., the obtained solutions can guarantee Pareto optimality, while B-MOMS cannot. In the experiments, we use multi-objective graph-coloring instances [12] and show that our proposed algorithm outperforms the state-of-the-art complete algorithm. We also compare the performances of our incomplete algorithm with the previous approximation algorithm and show that we can provide much better solutions but require more time, offering an interesting trade-off between speed and quality.

The rest of the section is organized as follows. In subsection 3.1.1, DPOP and its operations are introduced. In subsection 3.1.2, we present MO-DPOP, discuss the theoretical complexity of the algorithm, and propose a technique to reduce the size of its messages while still guaranteeing to find a subset of the Pareto front. In subsection 3.1.3, we evaluate the performances of MO-DPOP against existing MO-DCOP algorithms. We finally conclude in subsection 3.1.4.

Algorithm 1 *Dynamic Programming Optimization Protocol for an agent* x_i

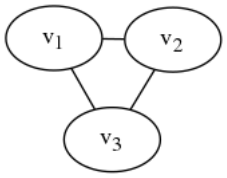
Phase 1: UTIL message propagation
$JOIN_i^{P_i} = \emptyset$
$JOIN_i^{P_i} = JOIN_i^{P_i} \oplus \left(\bigoplus_{C_k \subseteq P_i \cup PP_i} R_k\right)$
for $x_j \in CH_i$ **do**
    receive $UTIL_j^i$ from x_j
    $JOIN_i^{P_i} = JOIN_i^{P_i} \oplus UTIL_j^i$
end for
$UTIL_i^{P_i} = JOIN_i^{P_i} \perp_{v_i}$
if $P_i \neq null$ **then**
    send $UTIL_i^{P_i}$ to P_i
end if
Phase 2: VALUE message propagation TODO
if $P_i = null$ **then**
    $val_i \leftarrow \underset{val_i \in D_i}{\operatorname{argmin}} JOIN_i^{P_i}[]$
else
    wait for $VALUES_{P_i}$ from parent
    $val_i \leftarrow \underset{val_i \in D_i}{\operatorname{argmin}} JOIN_i^{P_i}[VALUES_{P_i} \cup val_i]$
end if
for each $x_j \in CH_i$ **do**
    send $VALUES_{P_i} \cup val_i$ to x_j
end for

3.1.1 DPOP

The Dynamic Programming Optimization Protocol (DPOP) [55] is the representative dynamic programming algorithm for Distributed Constraint Optimizaton Problems. We will present the algorithm and the different messages and operations it uses to help understand our extension for multiple objectives that we will present in Section 3.1.2. The algorithm is made of three phases. The first phase organizes the agents into a pseudo-tree as discussed in Section 2.3.4. The second phase, called UTIL phase, computes the best utility of each sub-tree in the pseudo-tree, which ends up producing the best global utility for the root agent. The third and last phase, called VALUE phase, communicates the resulting solutions to all agents.

Algorithm 1 presents the pseudo-code of DPOP and assumes that agents are already organized into a pseudo-tree structure such that each agent x_i is aware of the following:

- P_i: the agent that is parent of x_i.
- PP_i: the set of agents that are pseudo-parents of x_i.
- CH_i: the set of agents that are children of x_i.
- Sep_i: the separator of x_i, i.e., the set of agents that are pseudo-parents of x_i, P_i or of any of its pseudo-parents PP_i.



v_1	v_2	$cost$	v_2	v_3	$cost$	v_1	v_3	$cost$
a	a	5	a	a	0	a	a	1
a	b	7	a	b	2	a	b	1
b	a	10	b	a	0	b	a	0
b	b	12	b	b	2	b	b	3

FIGURE 3.1: Example of bi-objective DCOP.

Most of the computation is performed during the UTIL phase of the algorithm, requiring to define a several operations used to compute the best utilities of a sub-tree.

In the first phase of DPOP, *UTIL* messages carrying the best utilities of the agent's sub-tree are sent up the pseudo-tree, starting from the leaf agents.

Definition 3.1 (*UTIL* message)**.** A message UTIL_i^j sent from agent x_i to agent x_j is a multi-dimensional matrix with one dimension for each variable in Sep_i and we write $dim(\text{UTIL}_i^j)$ the set of variables considered by the message.

This message expresses the best utilities that can be obtained by the sub-tree rooted at x_i based on the values taken by the variables in the separator Sep_i. We note that a cost function $f_k : \times_{\forall v_i \in C_k} D_i \rightarrow \mathbb{R}_{\geq 0}$ can be represented as a matrix whose dimensions are the variables included in the constraint, i.e., $dim(f_j) = C_j$.

Example 3.1 (Multi-dimensional representation)**.** *The cost function of the constraint* $\{v_1, v_2\}$ *shown in Figure 3.1 can be represented as a matrix* $U_{1,2}$ *as follows:*

$v_1 \setminus v_2$	a	b
a	5	7
b	10	12

Definition 3.2 (Slice)**.** Given a multi-dimensional matrix M and an assignment $A \subseteq dim(\text{M})$, a slice of M along A, written $M[A]$, is a matrix of dimension $\{dim(\text{M}) \setminus A\}$ such that:

$$\text{M}[A] = \bigcup_{\forall A', A' = dim(\text{M}) \setminus A} \text{M}[A \cup A']$$

Example 3.2 (Slice)**.** *Using the matrix* $M_{1,2}$ *shown in example 3.1, we can consider the following slices:*

- $M_{1,2}[\{v_1 = a\}] =$

v_2	a	b
	5	7

.

- $M_{1,2}[\{v_1 = b, v_2 = b\}] = 12$.

The *UTIL* message $UTIL_i^j$ sent by an agent x_i to its parent $P_i = x_j$ is the combination of:

- All $UTIL_k^i$ messages received by x_i from one of its children $x_j \in CH_j$.

- The matrix representations of cost functions between x_i and its parents and pseudo-parents.

We now define the operation used to combine these matrices.

Definition 3.3 (Join Operator)**.** Joining two matrices M and M', written $M \oplus M'$, produces a new matrix M'' such that $dim(M'') = dim(M) \cup dim(M')$ and:

$$\forall A = dim(M''), M''[A] = M[A] + M'[A]$$

Example 3.3 (Join Operator)**.** *Joining two matrices of dimension* $\{v_1\}$ *and* $\{v_2\}$ *results in a matrix of dimension* $\{v_1, v_2\}$ *as follows:*

v_1	*a*	*b*
	1	*2*

$\oplus$

v_2	*a*	*b*
	3	*4*

$=$

$v_1 \setminus v_2$	*a*	*b*
a	*4*	*5*
b	*5*	*6*

Joining two matrices of dimension $\{v_1, v_2\}$ *and* $\{v_2\}$ *results in a matrix of dimension* $\{v_1, v_2\}$ *as follows:*

$v_1 \setminus v_2$	*a*	*b*
a	*4*	*5*
b	*5*	*6*

$\oplus$

v_2	*a*	*b*
	1	*2*

$=$

$v_1 \setminus v_2$	*a*	*b*
a	*5*	*7*
b	*6*	*8*

Before sending an *UTIL* message, an agent x_i simplifies the matrix obtained using the join operator by projecting its own variable v_i out of the matrix, removing the variable from the matrix' dimension.

Definition 3.4 (Projection Operator)**.** Projecting variable v_i out of matrix M, written $M \perp_{v_i}$ and requiring $v_i \in dim(M)$, is the projection through optimization of the matrix M along the v_i dimension:

$$\forall A = \{dim(M) \setminus v_i\}, M \perp_{v_i}[A] = \min_{v_i} M[A \cup \{v_i\}]$$

Example 3.4 (Projection Operator)**.** *Projecting variable* v_1 *out of a matrix of dimension* $\{v_1, v_2\}$ *results in a new matrix of dimension* $\{v_2\}$ *as follows:*

$v_1 \setminus v_2$	*a*	*b*
a	*1*	*4*
b	*3*	*2*

$\perp_{v_2} =$

v_2	*a*	*b*
	1	*2*

Since each agent projects its own variable out of the *UTIL* message before sending it, and since the relations added to a message cannot include variables from other branches (due to the pseudo-tree structure), the *UTIL* messages received by the root x_r are all matrices with one dimension $\{v_r\}$. This allows the root agent to compute its local value that produces an optimal solution.

In the second phase of DPOP, *VALUE* messages containing partial assignments are sent down the pseudo-tree, propagating the optimal solution of the problem to all agents.

3.1.2 MO-DPOP

In this subsection, we present the Multi-Objective Distributed Pseudo-tree Optimization Algorithm (MO-DPOP). We first show how to extend the operations used in DPOP to the multi-objective case before presenting the pseudo-code of the algorithm and discussing its properties.

3.1.2.1 Extensions of Operators

MO-DPOP extends DPOP in the sense that elements of the matrices carried by *UTIL* messages are now vectors from $\mathbb{R}^m_{\geq 0}$ instead of single values from $\mathbb{R}_{\geq 0}$. This requires to define new operators for the addition of vectors and set of vectors.

Definition 3.5 (Direct sum)**.** Given two vectors $U = (u_1, .., u_n)$ and $W = (w_1, .., w_n)$, $U \oplus W = (u_1 + w_1, .., u_n + w_n)$ is the direct sum of U and W.

Definition 3.6 (Set addition)**.** Given two sets of vectors $\mathcal{U} = \{U_1, \ldots, U_k\}$ and $\mathcal{W} = \{W_1, \ldots, W_l\}$ then $\mathcal{U} \uplus \mathcal{W} = \{U \oplus W, U \in \mathcal{U}, W \in \mathcal{W}\}$ is the set addition of $\mathcal{U}$ and $\mathcal{W}$.

Next, we extend the *join* and *projection* operators that are used to construct *UTIL* messages. Joining two matrices with vector elements is straightforward now that we have defined set addition.

Definition 3.7 (Join Operator in MO-DPOP)**.** Joining two matrices M and M', written $M \oplus M'$, produces a new matrix M'' such that $dim(M'') = dim(M) \cup dim(M')$ and:

$$\forall A = dim(M''), M''[A] = M[A] \uplus M'[A]$$

The projection of a variable out of a matrix with vector elements requires to filter dominated solutions, where a simple minimization was sufficient in the mono-objective case. For simplicity, we consider a function $ND(\mathcal{U}) = \{U \in \mathcal{U} | \nexists U' \in \mathcal{U}\ s.t.\ U' \prec U\}$which takes a set of vectors $\mathcal{U}$ and returns the corresponding set of non-dominated vectors.

Definition 3.8 (Projection Operator in MO-DPOP)**.** Projecting variable v_i out of matrix M, written $U \perp_{v_i}$ and requiring $v_i \in dim(M)$, is the projection through optimization of the matrix M along the v_i dimension:

$$\forall A = \{dim(M) \setminus \{v_i\}\}, M \perp_{v_i}[A] = \text{ND}(M[A])$$

Example 3.5 (Operators in MO-DPOP)**.** *We consider the matrices R_{23} and R_{13} representing the cost functions of constraints $\{v_2, v_3\}$ and $\{v_1, v_3\}$ of Figure 3.1. Table 3.1 shows the result of the join operator ($\oplus$) applied on R_{23} and R_{13}. Table 3.2 shows the result of projecting variable v_3 out of the matrix obtained from the previous join.*

v_1	v_2	v_3	cost
a	a	a	(1,1)
a	a	b	(3,1)
a	b	a	(1,2)
a	b	b	(3,0)
b	a	a	(0,2)
b	a	b	(5,3)
b	b	a	(0,3)
b	b	b	(5,2)

TABLE 3.1: $R_{23} \oplus R_{13}$.

v_1	v_2	costs
a	a	(1,1)
a	b	(1,2) (3,0)
b	a	(0,2)
b	b	(0,3) (5,2)

TABLE 3.2: $(R_{23} \oplus R_{13}) \perp_{v_3}$.

3.1.2.2 Algorithm

Algorithm 2 shows the pseudo-code of the MO-DPOP algorithm. We assume that the generation of the pseudo-tree was done in a preprocessing step such that each agent x_i is aware of its parent P_i, its pseudo-parents PP_i, and its children CH_i. We do not detail the *VALUE* phase as it is similar to the mono-objective case.

The construction of the *UTIL* message that x_i sends to its parent P_i starts with the local cost functions of the agent. First, the cost functions of constraints involving v_i and its parent or pseudo-parents are turned into matrices and joined together (line 2). For simplicity, we denote R_k the matrix representing a set of cost functions $\{f_k^1, \ldots, f_k^m\}$ corresponding to constraint C_k. After adding local cost functions, each matrix received from a child is also added (line 5). Then, the agent projects its own variable out of the matrix to obtain the message $UTIL_i^{P_i}$ (line 7) that it sends to its parent (line 9). Leaf agents are responsible for starting the *UTIL* phase by sending the first messages. The root is the last agent to receive *UTIL* messages and is then responsible for selecting a solution from the Pareto front before starting the *VALUE* phase.

Algorithm 2 *MO-DPOP - UTIL Phase of agent* x_i

1: $JOIN_i^{P_i} = \emptyset$
2: $JOIN_i^{P_i} = JOIN_i^{P_i} \oplus \left(\bigoplus_{C_k \subseteq P_i \cup PP_i \cup \{v_i\}} R_k \right)$
3: **for** $x_j \in CH_i$ **do**
4: receive $UTIL_j^i$ from x_j
5: $JOIN_i^{P_i} = JOIN_i^{P_i} \oplus UTIL_j^i$
6: **end for**
7: $UTIL_i^{P_i} = JOIN_i^{P_i} \perp_{v_i}$
8: **if** $P_i \neq null$ **then**
9: send $UTIL_i^{P_i}$ to P_i
10: **end if**

3.1.2.3 Properties

Similarly to the mono-objective case, MO-DPOP uses a linear number of messages and *UTIL* messages are exponential in size.

Property 1 (Number of Messages). Since every agent, root excepted, sends one UTIL message and receives one VALUE message, the total number of messages required is $2(n-1)$.

Property 2 (Maximum Message Size). UTIL messages have a space complexity of $O(m \times |D_{max}|^n)$ where $|D_{max}|$ is the maximum domain size in the problem.

Proof. In the worst case, each possible assignment produces a non-dominated vector of size m, causing the projection operator to be unable to remove any vector during the optimization process. In the case where the pseudo-tree is a chain, the last *UTIL* message sent to the root x_r contains $\prod_{D \in \mathcal{D} \setminus D_r} D$ vectors. □

In the mono-objective case however, the size of an *UTIL* message sent by an agent x_i is bounded by the size of its separator Sep_i. This is due to the strict ordering when comparing single values, guaranteeing that the optimization phase of the projection yields a single value. Next we will show how to obtain a similar complexity in the multi-objective case.

3.1.2.4 Limiting the Size of Messages

We now propose MO-DPOP(B_b^Ω), an incomplete version of our algorithm where *UTIL* messages are bounded in size by the separators of the agents. This bounding is done using a function B_b which takes a set of vectors $\mathcal{U}$ and returns a set $\mathcal{W} \subseteq \mathcal{U}, s.t. |\mathcal{W}| \leq b$. The idea is to apply this function to each element of the multi-dimensional matrix to limit the size of *UTIL* messages.

Since a bounding function B_b always selects a subset of maximum size b, we can guarantee the maximum size of an *UTIL* message.

Property 3 (Maximum Bounded Message Size). If the subset $\mathcal{U}$ yielded by B is bounded in size ($|\mathcal{U}| \leq b$), the maximum message size becomes bounded by the maximum separator size $|\text{Sep}_{max}|$ with a space complexity in $O(bm \times |D_{max}|^{|\text{Sep}_{max}|})$.

Proof. An *UTIL* message sent by an agent x_i contains $\prod_{x_k \in Sep_i} |D_k|$ elements. In the multi-objective case, an element is a set of vectors $\mathcal{U}$, each vector being of length m. If this set is bounded by b ($|\mathcal{U}| \leq b$), the size of the message can never be more than $bm \times \prod_{x_k \in Sep_i} D_k$, giving a space complexity of $O(bm \times |D_{max}|^{|Sep_{max}|})$ based on the maximum domain size D_{max} and maximum separator size Sep_{max} □

If we use a function B_1 that always returns a single vector, the space complexity is equivalent to that of the original DPOP algorithm. Such a bounding function can be defined using scalarization techniques such as the weighted-sum that are known to guarantee Pareto optimality [56].

Definition 3.9 (Weighted-Sum Selection)**.** Given a set $\mathcal{S}$ of vectors of size m and a weight $\omega = (\omega_1, \ldots, \omega_m)$ such that $0 < \omega_i \leq 1$ and $\sum_{\omega_i \in \omega} \omega_i = 1$, a weighted-sum selection $B_1^{\omega}(\mathcal{S})$ yields the vector $S^* = \underset{S \in \mathcal{S}}{\operatorname{argmin}} \sum_{i}^{m} \omega_i S_i$.

In this paper, we propose to use a bounding function B_b^{Ω} that makes use of weighted-sum techniques. By providing a set of weights Ω, $|\Omega| = b$, the function returns a set $\mathcal{U}$ containing one vector for each weight. This function is applied to filter each element of the *UTIL* matrix before it is sent to the parent agent. In the root agent, the function can be applied to the final set of solutions to extract a subset of guaranteed Pareto optimal solutions.

Property 4. In the root agent x_r, the vectors returned by the function B_b^{Ω} applied on UTIL_r^{null} are all Pareto optimal solutions.

Proof. When using a bounding function B_b^{Ω} during the *UTIL* phase of MO-DPOP, *UTIL* messages contain, for each combination of values of their variables, the vectors that optimize the weighted-sum for a weight $\omega \in \Omega$. This guarantees that combining *UTIL* messages at the root agent will yield vectors that also optimize a weighted-sum. Since a weighted-sum solution is Pareto optimal, the solutions obtained at the root using the bounding function are all Pareto optimal. □

Example 3.6 (Weighted-Sum Selection)**.** *Given a set* $\mathcal{S} = \{(1,7),(3,4),(4,2),(5,0)\}$*, and a set of weights* $\Omega = \{(0.5,0.5),(0.9,0.1)\}$*, the result of using function* B_2^{Ω} *on* $\mathcal{S}$ *is* $\{(1,7),(5,0)\}$*.* $(5,0)$ *minimizes the weighted-sum when using* $\omega = (0.5,0.5)$ *and* $(1,7)$ *minimizes the weighted-sum when using* $\omega = (0.9,0.1)$*.*

Using a weighted-sum selection to filter the *UTIL* messages of MO-DPOP *guarantees* to find at least one Pareto optimal solution. However, the more weights are provided (corresponding to higher values of b), the more Pareto optimal solutions we can expect to find. In addition, using a weighted-sum selection still lets us find solutions that do not optimize any of the weighted-sum provided, due to the root agent not having to filter its own *UTIL* matrix. By guaranteeing a subset of the Pareto front, the weighted-sum selection provides an incomplete method for MO-DCOPs.

Several other bounding functions can be considered to limit the size of each UTIL message while still guaranteeing Pareto optimality. For example, we could consider functions based on different orderings of the objectives, known as lexicographic ordering [57], or functions based on a number of predefined reference points [58].

3.1.3 Experimental Evaluation

In our experiments, we evaluate our complete algorithm MO-DPOP, its incomplete version with bounded messages MO-DPOP(B_b^Ω), and the state-of-the-art complete algorithm MO-ADOPT. We start by presenting the settings used for our evaluation. Then, we will compare the performances of MO-DPOP with MO-ADOPT. Finally, we will evaluate our incomplete method for various values of the parameter b.

3.1.3.1 Experimental setup

For our experiments, we use an extended graph-coloring problem [12] which was used to evaluate the performances of previous MO-DCOP algorithms. Graph-coloring is used in real applications such as scheduling problems [50] and wireless sensor networks [36]. In this problem, each agent x_i owns a variable v_i taking its value from the domain $D_i = \{1, 2, 3\}$ and the following three cost functions are considered:

$$f^1(v_i, v_j) = \begin{cases} 0 & v_i \neq v_j \\ 1 & v_i = v_j \end{cases}, f^2(v_i, v_j) = \begin{cases} 0 & \text{if } |v_i - v_j| = 1 \\ 1 & \text{otherwise} \end{cases},$$

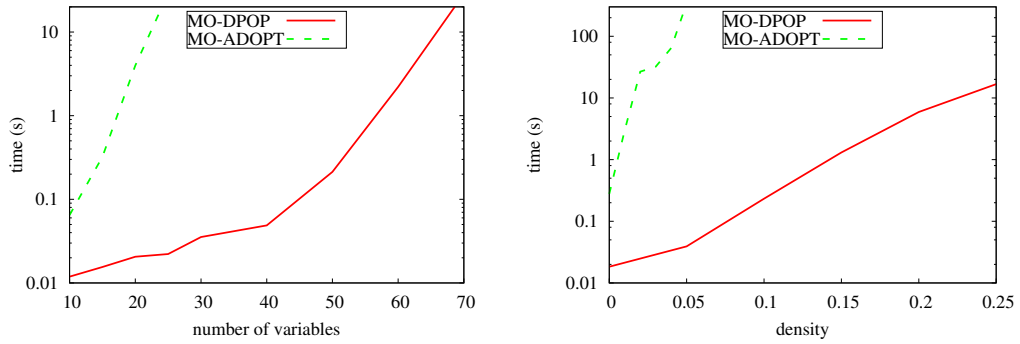
$$f^3(v_i, v_j) = \begin{cases} 0 & \text{if } i < j \text{ and } v_i < v_j \\ 1 & \text{otherwise} \end{cases}$$

The first function represents the common graph coloring conflict function. The aim of the second function is to have a distance of one between the values of neighboring variables. The last function considers that variables with higher indexes should take higher values.

Our problem instances are created using random connected graphs generated using a specific number of variables (n) and a specific density (δ).When $\delta = 0$, the number of edges in the graph is equal to $n-1$ and when $\delta = 1$, the number of edges is equal to $\frac{n}{2}(n-1)$. All results shown are averages over 40 instances. All algorithms were implemented in Java and all experiments were carried on a 4.2 GHz 8 cores CPU with 8GB of RAM dedicated to the Java Virtual Machine and using a timeout of 10 minutes.

3.1.3.2 Results - Complete Approach

We first compare the performance of our algorithm with MO-ADOPT [30], the previous complete MO-DCOP algorithm. Figure 3.2(a) shows the runtime of both algorithms when varying the number of variables on a sparse problem (δ), and Figure 3.2(b) shows the runtime when varying the density of problem with 20 variables. We observe that both algorithms are time-exponential in terms of both variable and density. While this is expected when solving an NP-hard problem, we notice that the runtime of MO-ADOPT increases at a much higher rate than



(a) varying variables ($\delta = 0.01$) (b) varying density ($n = 20$)

FIGURE 3.2: Runtime comparison between MO-DPOP and MO-ADOPT.

MO-DPOP. When varying the number of variables, we see that MO-ADOPT needs 28 seconds when $n = 25$ but times out (10 minutes or more) for $n \geq 30$. Meanwhile, MO-DPOP can solve problems of up to 70 variables in less than 30 seconds and takes only 22 milliseconds for $n = 25$. When varying the density, MO-ADOPT times out for instances where $\delta > 0.05$ while we could solve instances with $\delta = 0.25$ in less than 20 seconds by using MO-DPOP.

Table 3.3 and Table 3.4 show the runtime, number of messages transmitted ($\#Msg$), and number of vectors exchanged ($\#Vectors$) of both algorithm. We also report the size of the Pareto front ($|PF|$) for the problems solved. Regarding the number of messages used by each algorithm, we notice an exponential increase for MO-ADOPT when varying the number of variables, going from 233 messages with $n = 10$ to 472600 with $n = 25$. Similarly, when the density increases, the number of messages used by MO-ADOPT goes from 3158 for $\delta = 0$ to 4098030 for $\delta = 0.05$. This high number of messages used is due to the search strategy of MO-ADOPT which uses many backtracks, requiring each time to inform many agents about the new solution space being explored. In comparison, MO-DPOP uses a linear number of messages ($n - 1$) that is not affected by the increase in variables or density. When looking at the number of vectors exchanged between the agents, we see that even though MO-DPOP uses a linear number of messages, their size increases exponentially. It thus communicates a lot of information for more difficult instances, exchanging 7219247 vectors for $n = 70$ and 1846101 for $\delta = 0.20$. As the problems grow larger, so does the separator sizes of the agents, which exponentially increases the dimensions of the MO-DPOP messages. For MO-ADOPT, its exponential number of messages results in a very high number of vectors exchanged as well. Thus, despite MO-DPOP using very large messages, it consistently communicates less vectors than MO-ADOPT, transmitting only 191 vectors for $n = 25$ and $\delta = 0.01$ compared to the 725794 used by MO-ADOPT.

These results show that MO-DPOP clearly outperforms MO-ADOPT, consistently requiring less time, using less messages, and exchanging less information. This seems to indicate that dynamic programming-based algorithms are better suited than search-based algorithms for solving

n	algorithm	t (s)	#Msg	#Vectors	$\|PF\|$
10	MO-ADOPT	0.065	233	250	1.5
	MO-DPOP	0.012	9	28	
20	MO-ADOPT	3.980	88823	104814	2.0
	MO-DPOP	0.020	19	140	
25	MO-ADOPT	28.86	472600	725794	2.6
	MO-DPOP	0.022	24	191	
30	MO-ADOPT	TO	-	-	3.0
	MO-DPOP	0.035	29	558	
40	MO-ADOPT	TO	-	-	4.6
	MO-DPOP	0.049	39	1710	
50	MO-ADOPT	TO	-	-	7.2
	MO-DPOP	0.21	49	44156	
60	MO-ADOPT	TO	-	-	11.6
	MO-DPOP	2.21	59	419148	
70	MO-ADOPT	TO	-	-	16.0
	MO-DPOP	28.59	69	7219247	

TABLE 3.3: Results varying variables ($\delta = 0.01$)

δ	algorithm	t (s)	#Msg	#Vectors	$\|PF\|$
0.00	MO-ADOPT	0.28	3158	3724	1.6
	MO-DPOP	0.02	19	62	
0.05	MO-ADOPT	342.32	4098030	5739400	4.1
	MO-DPOP	0.04	19	2393	
0.10	MO-ADOPT	TO	-	-	8.19
	MO-DPOP	0.23	19	48636	
0.20	MO-ADOPT	TO	-	-	15.7
	MO-DPOP	5.92	19	1846101	

TABLE 3.4: Results varying density ($n = 20$)

MO-DCOPs. In the mono-objective case, both search-based and dynamic programming-based approaches are viable, with search being slower but requiring only a linear or polynomial space complexity. In the multi-objective case however, this advantage is negated by the exponential nature of the Pareto front as well as the difficulty to perform pruning with multiple objectives. This makes dynamic programming-based algorithms such as MO-DPOP better suited for tackling MO-DCOPs.

3.1.3.3 Results - Incomplete Approach

We now evaluate the impact of using a bounding function, as proposed in subsection 3.1.2.4, to limit the size of the messages exchanged during our algorithm. For this experiment, we use a weighted-sum selection B_b^Ω and vary the adjustable parameter b between 1 and 5, letting us adjust how much the messages of MO-DPOP are filtered.

variables	density	algorithm	t (s)	#Vectors	$\|Sols\|$	$\%PF$	$\%PO$
20	0.1	MO-DPOP	0.23	48636	8.2	100%	100%
		MO-DPOP(B_1^Ω)	0.14	11882	2.0	22%	77%
		MO-DPOP(B_2^Ω)	0.16	17994	3.2	35%	79%
		MO-DPOP(B_3^Ω)	0.20	28648	4.1	50%	88%
		MO-DPOP(B_4^Ω)	0.20	29769	4.5	53%	87%
		MO-DPOP(B_5^Ω)	0.21	32417	5.0	59%	88%
20	0.2	MO-DPOP	5.92	1846101	15.7	100%	100%
		MO-DPOP(B_1^Ω)	2.38	363742	2.1	12%	80%
		MO-DPOP(B_2^Ω)	2.98	589919	3.5	20%	78%
		MO-DPOP(B_3^Ω)	4.04	952687	4.7	29%	84%
		MO-DPOP(B_4^Ω)	4.25	1011791	5.4	33%	84%
		MO-DPOP(B_5^Ω)	4.66	1137181	6.3	38%	85%
20	0.3	MO-DPOP	TO	-	-	-	-
		MO-DPOP(B_1^Ω)	18.99	2626643	2.3	-	-
		MO-DPOP(B_2^Ω)	24.86	4314520	3.5	-	-
		MO-DPOP(B_3^Ω)	40.08	7202826	4.6	-	-
		MO-DPOP(B_4^Ω)	43.06	7873157	5.8	-	-
		MO-DPOP(B_5^Ω)	47.14	8728822	6.8	-	-

TABLE 3.5: Impact of different bounding functions using various settings.

Since we now consider incomplete methods, the set of solutions found by our algorithm (denoted $Soll$) might differ from the actual Pareto front (PF) of the problem. To evaluate the quality of the solutions found, we use the following additional measures: $|Sols|$, the number of solutions found by the algorithms (not all of these solutions are actually Pareto optimal); $\%PF$, the ratio of the actual Pareto front that was found by the algorithm, i.e., $\frac{|PF \cap Sols|}{|PF|}$; $\%PO$, the ratio of solutions found by the algorithm that are actually Pareto optimal, i.e., $\frac{|PF \cap Sols|}{|Sol|}$.

Table 3.5 shows the results obtained using our incomplete algorithm on instances of 20 variables and with various densities ($\delta \in \{0.1, 0.2, 0.3\}$). For each setting, we first show the results of our complete algorithm, which always finds the complete Pareto front (100% of the Pareto front is found and 100% of the solutions found are Pareto optimal). For the harder setting ($\delta = 0.3$), our complete approach requires too much memory and we thus could not compute the exact Pareto front.

We first notice that the number of solutions found ($|Sols|$) is always higher or equal to the number of weights (b) given to our incomplete method. Some of these solutions are guaranteed to be Pareto optimal as they optimize at least one of the weighted-sum provided. This leads to most of the solutions being Pareto optimal ($77\% \leq \%PO \leq 88\%$), indicating that the weighted-sum selection method is suitable to target a subset of the Pareto front. Regarding the impact of b on the number of vectors exchanged between the agents, we see a large reduction of the communication cost when b is low. For $\delta = 0.2$ and $b = 1$, the number of vectors is around 20% of what it is for our complete approach. However, by filtering the messages, we also miss

on some Pareto optimal solutions, finding only subsets of the Pareto front (for $\delta = 0.2$, we find 12% of the Pareto front for $b = 1$, and 38% for $b = 5$).

These results show that the weighted-sum selection is a good approach to reduce the communication cost of MO-DPOP while still providing a subset of the Pareto front. Our incomplete method consistently finds at least as many solutions as weights provided and we observed that most of these solutions are actual Pareto optimal solutions. Bounding the size of messages thus offers a trade-off between solution quality and performances, reducing the runtime and communication cost but also reducing the amount of Pareto optimal solutions found. The main impact of the adjustable parameter b being on the number of vectors exchanged between agents, this can help use our algorithm in real multi-agent systems where communications can be very costly. Most importantly, our incomplete algorithm offers the advantage of *guaranteeing* some Pareto optimal solutions, a guarantee that was missing from the previous approximation algorithm for MO-DCOP.

3.1.4 Conclusion

The Multi-Objective Distributed Constraint Optimization Problem (MO-DCOP) can model many problems from multi-agent systems and some applications might require to provide complete and exact solutions. However, few algorithms exist for MO-DCOPs with only one complete algorithm proposed so far. In this paper, we developed a new complete algorithm for MO-DCOP as well as an incomplete extension that reduces the complexity of our algorithm but still provides a subset of the Pareto front. Our experiment showed that our complete algorithm outperforms the state-of-the-art complete MO-DCOP algorithm and that our incomplete approach offers better solutions than the previous approximation algorithm but requires more time.

In future works, we will study additional ways to reduce the complexity of our algorithm by considering techniques that have been previously proposed in the centralized case with the Mini-Bucket Elimination [29] or in the mono-objective distributed case with the Memory-Bounded DPOP [59] and the p-reduced graph technique [60]. We will also study the different subsets that can be obtained using various message bounding functions as specific subsets of the Pareto front can be considered more interesting than others [61, 62].

3.2 Distributed Pareto Local Search

Since in a multi-objective problem all assignments can produce non-dominated solutions, finding the complete set of Pareto optimal solutions becomes easily intractable for large-scale instances. This is why it is important to study fast approaches that can approximate the Pareto front. Moreover, using MO-COP techniques in a distributed setting would require that one agent gathers all information about the problem and performs all computations. However, agents in a distributed system generally care about their privacy and are limited in their communication and computation capabilities, thus requiring approximation MO-DCOP algorithms.

In this section, we develop a novel approximation algorithm called *Distributed Pareto Local Search* (DPLS) algorithm for solving an MO-DCOP. This algorithm is the extension of the well-known Pareto Local Search (PLS) [63] designed for approximating the Pareto front of multi-objective optimization problems. PLS is the generalization of the hill-climbing method for optimization problems with multiple criteria. With the DPLS, we propose an extension of this method for MO-DCOPs. In the experiments, we evaluate the performances of DPLS with different problem settings and show that the local search technique is suitable for solving an MO-DCOP. We also compare DPLS with the state-of-the-art approximation MO-DCOP algorithm B-MOMS, and empirically show that our proposed algorithm DPLS outperforms the state-of-the-art B-MOMS.

The rest of the section is organized as follows. In subsection 3.2.0.1, we introduce the Pareto Local Search, a centralized MO-COP algorithm. In subsection 3.2.1, we present a new approximation algorithm for MO-DCOPs which extends the Pareto Local Search to a distributed environment. In subsection 3.2.2, we evaluate the performance of our proposed algorithm on instances of multi-objective graph-coloring problem and compare these results with the state-of-the-art approximation algorithm for MO-DCOPs. We then conclude in subsection 3.2.3.

Algorithm 3 Pareto Local Search

1: **Input**: $\mathcal{A}_0$ an initial set of assignments
2: $\mathcal{A} \leftarrow ND(\mathcal{A}_0)$
3: $archive \leftarrow \mathcal{A}$
4: **while** $archive \neq \emptyset$ **do**
5: $A \leftarrow$ random assignment in $archive$
6: **for** each $A' \in \mathcal{N}(A)$ **do**
7: $\mathcal{A} \leftarrow ND(\mathcal{A} \cup \{A'\})$
8: **end for**
9: explored(A) $\leftarrow$ true
10: $archive \leftarrow \{A \in \mathcal{A} | \neg explored(A)\}$
11: **end while**
12: **Output**: $\mathcal{A}$, an approximation of the Pareto front

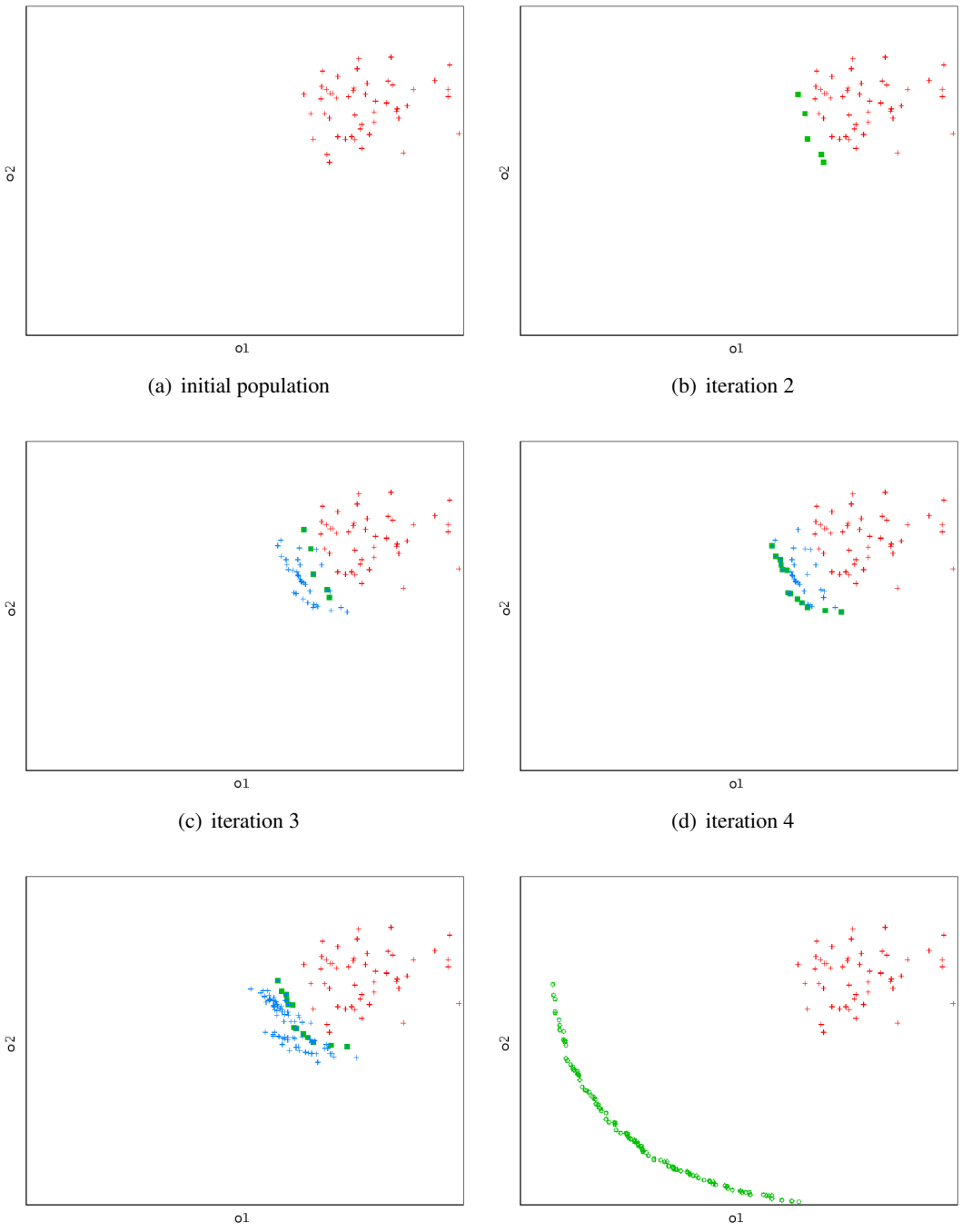
3.2.0.1 Pareto Local search

The idea of a local search is to iteratively improve a solution by exploring its neighborhood. The best solution within this neighborhood is kept and its own neighborhood is in turn explored. This is repeated until no improvement can be found in the neighborhood of the current best solution. In the mono-objective case, a total ordering of the solutions exists, making the selection of the improving solution straight-forward. However, in the multi-objective case, multiple solutions can offer an improvement when using the dominance relation, requiring new dominance-based algorithms.

For centralized Multi-Objective Optimization Problems, the Pareto Local Search (PLS) [63] was proposed as an extension of the standard local search from the mono-objective case to the multi-objective case. Algorithm 3 shows the PLS framework, which can be used to centrally compute the Pareto front of an MO-DCOP. PLS takes as input an initial set of assignments $\mathcal{A}_0$ which forms the initial set of non-dominated assignments $\mathcal{A}$ (line 2) to be returned at the end of the search. At each iteration, an assignment A is randomly taken from the $archive$ (line 5), the set of assignments in $\mathcal{A}$ not yet explored. Using an operator ND, which returns all the non-dominated solutions in a given set, we add all non-dominated neighbors of A to $\mathcal{A}$ (line 7) and consider A to now be explored. PLS terminates once the $archive$ is empty, meaning that all non-dominated assignments in $\mathcal{A}$ have been explored. Figure 3.3 shows an example of multiple iterations of PLS.

Similarly to a mono-objective local search, PLS requires to define a neighborhood operator $\mathcal{N}(A)$ (line 6) that generates the neighborhood of A by applying small changes to it. Common neighborhood operators change the value of one variable at a time or swaps values between two variables. Compared to a mono-objective local search where only one local optimal solution need to be considered at a time, PLS requires to maintain a set of locally optimal solutions (denoted by $\mathcal{A}$ in the algorithm).

In the next subsection, we propose to distribute the PLS in order to compute the Pareto front of MO-DCOPs in a distributed fashion.



(a) initial population (b) iteration 2

(c) iteration 3 (d) iteration 4

(e) iteration 5 (f) Approximation of Pareto front

FIGURE 3.3: Behavior of DPLS. Red crosses represent the initial solutions generated using Algorithm 4 (a). Green squares show the non-dominated subset of the initial solutions used as input of DPLS (b). New red crosses represent the neighbors of the green squares (c). DPLS keeps exploring the neighbors of non-dominated solutions (d,e). After a finite number of iteration, we obtain an approximation of Pareto front (f).

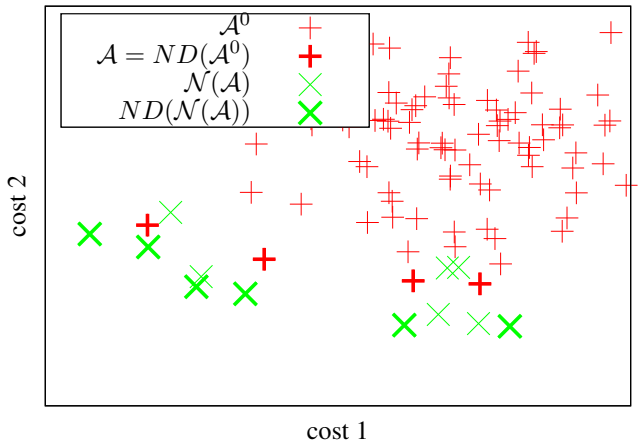


FIGURE 3.4: Example of the first iteration of DPLS

3.2.1 Distributed Pareto Local Search Algorithm

In this subsection, we propose a novel approximation algorithm for MO-DCOPs called *Distributed Pareto Local Search* (DPLS). This algorithm extends the Pareto Local Search (PLS) [63] presented in subsection 3.2.0.1, from a centralized to a distributed setting.

To adapt the PLS for distributed settings, we assume that: (i) an agent only knows the cost functions involving its own variable, and (ii) two agents can only communicate if a constraint exists between their variables. To comply with the first assumption, we suppose that an agent x_i only knows its local cost functions which include variable v_i. To comply with the second assumption, agents are ordered in a pseudo-tree, a popular graph structure for DCOP algorithms [26]. In a pseudo-tree, all variables sharing a constraint are required to be part of a same path between the root and a leaf. Such structure can be obtained using a depth-first traversal of the constraint hypergraph. Each agent x_i can only send or receive messages to or from its parent $parent_i$, and children $children_i$, in the pseudo-tree.

The DPLS has two phases. In the first phase, an initial set $\mathcal{A}^0$ of solutions is generated. In the second phase, a local search is performed in a distributed fashion, maintaining a set of non-dominated solutions $\mathcal{A}$, initialized using $\mathcal{A}^0$. At each iteration, the neighborhoods of some solutions in $\mathcal{A}$ are explored in an attempt to find new non-dominated solutions. Figure 3.4 shows an example of the first iteration of DPLS. From a random set of solutions ($\mathcal{A}^0$), only the non-dominated solutions $\mathcal{A}$ are considered. Their neighborhood $\mathcal{N}(\mathcal{A})$ is explored, offering new non-dominated solutions. To perform this search in a distributed way, each agent x_i uses a local operator $\mathcal{N}_i$ such that $\mathcal{N}_i(A)$ generates the set of assignments that differs from A only by the value of variable v_i. At each iteration, the root sends the assignments to explore down the pseudo-tree. Each agent then computes its corresponding local neighborhood and sends the new non-dominated solutions up the tree. To limit the number and size of messages, an

Algorithm 4 Distributed Generation of a Solution for x_i

1: $pA_i \leftarrow$ random value in D_i
2: $r_i \leftarrow \mathbf{0}$
3: **for** each $x_j \in children_i$ **do**
4: x_i receives (pA_j, r_j) from x_j
5: $pA_i \leftarrow pA_i \cup pA_j$
6: $r_i \leftarrow r_i \oplus r_j$
7: **end for**
8: $r_i \leftarrow r_i \oplus R_i(pA_i)$
9: send (pA_i, r_i) to $parent_i$
10: **Output**: a random solution pA (at root agent)

agent waits for the solutions from its children before sending its own neighborhood. In addition, it compares its own neighborhood with the ones received from its children and discards all solutions that are dominated. The iteration finishes once the root has received neighborhoods from all its children and updated the set of optimal solutions accordingly. The assignments explored at each iteration are randomly selected within the set of *unexplored* non-dominated solutions and a parameter e is used to adjust the number of solutions selected. Increasing this number can lead to faster convergence, decreasing the number of messages used by DPLS but increasing their size. This can be an important adjustment in distributed systems depending on the quality of the communication network. While the original PLS algorithm only considered a single unexplored solution per iteration (equivalent to $e = 1$), other works have considered all unexplored solutions [64] or a similar parameter to limit the exploration [65]. The search phase also requires for each agent x_i a neighborhood operator $\mathcal{N}_i$ to be defined.

3.2.1.1 Algorithm

Algorithm 4 shows the pseudo-code for generating a random solution. Starting from the leaf agents in the pseudo-tree, the idea of this algorithm is to propagate a partial assignment pA up the pseudo-tree, along with its cost. Each agent x_i starts by selecting a random value from its variable domain D_i (line 1) that is added to the partial assignments received from its children (line 5). A partial cost r_i is computed by adding the costs r_j received from the children (line 6) with the local cost generated by the partial assignment $R_i(pA)$ (line 8). Due to the requirement that variables sharing a constraint are part of a same path between the root of the pseudo-tree and a leaf agent, an agent x_i can compute a local cost vector $R_i(pA_i)$ corresponding to the costs generated by all the constraints involving x_i and its descendants. This partial cost, along with the partial assignment, is sent to the parent or, in the case of the root agent, the assignment is complete and we know its corresponding cost vector.

Algorithm 5 shows the pseudo-code of the search phase. Similarly to the centralized PLS, we maintain a set of non-dominated solutions $\mathcal{A}$ from which we consider $archive$, the set of solutions in $\mathcal{A}$ not yet explored. $\mathcal{A}$ and $archive$ are initialized with the set of non-dominated solutions generated using Algorithm 4. At each iteration, the root selects a subset $subarchive$

Algorithm 5 Distributed Pareto Local Search for x_i

1: **Input**: $\mathcal{A}_0$ an initial set of assignments; e the maximum number of assignments to explore at each iteration
2: $\mathcal{A} \leftarrow ND(\mathcal{A}_0)$
3: **if** x_i is root **then**
4: $archive \leftarrow \mathcal{A}$
5: $subarchive \leftarrow$ selectSubset($archive$, e)
6: send $(subarchive)$ to each $x_j \in children_i$
7: $neighbors_i \leftarrow ND(\mathcal{N}_\rangle(subarchive))$
8: **end if**
9: **while** $\neg terminated_i$ **do**
10: x_i receives message M
11: **if** $M = (terminate)$ **then**
12: $terminated_i \leftarrow true$
13: send $(terminate)$ to each $x_j \in children_i$
14: **end if**
15: **if** $M = (subarchive)$ **then**
16: send $(subarchive)$ to each $x_j \in children_i$
17: $neighbors_i \leftarrow ND(\mathcal{N}_i(subarchive))$
18: **if** x_i is leaf **then**
19: send $(neighbors_i)$ to $parent_i$
20: **end if**
21: **end if**
22: **if** $M = (neighbors_j)$ **then**
23: $neighbors_i \leftarrow neighbors_i \cup neighbors_j$
24: **if** x_i received all $neighbors_j, \forall x_j \in children_j$ **then**
25: $neighbors_i \leftarrow ND(neighbors_i)$
26: send $(neighbors_i)$ to $parent_i$
27: **if** x_i is root **then**
28: $explored(A) \leftarrow true, \forall A \in subarchive$
29: $\mathcal{A} \leftarrow ND(\mathcal{A} \cup neighbors_i)$
30: $archive \leftarrow \{A \in \mathcal{A} | \neg explored(A)\}$
31: **if** $archive = \emptyset$ **then**
32: $terminated_i \leftarrow true$
33: send $(terminate)$ to each $x_j \in children_i$
34: **else**
35: $subarchive \leftarrow$ selectSubset($archive$, e)
36: send $(subarchive)$ to each $x_j \in children_i$
37: $neighbors_i \leftarrow ND(\mathcal{N}_i(subarchive))$
38: **end if**
39: **end if**
40: **end if**
41: **end if**
42: **end while**
43: **Output**: $\mathcal{A}$, an approximation of the Pareto front

of size e from the $archive$ (line 5). This $subarchive$ is sent down the pseudo-tree and corresponding non-dominated neighbors are sent back up the tree. Upon reaching the root, the new non-dominated solutions are added to $\mathcal{A}$ and the $archive$ is updated accordingly. This is repeated until all solutions in $\mathcal{A}$ have been explored.

To perform the search, three messages are used. To propagate the assignments to explore, we use $subarchive$ messages (line 15). The search is started when the root agent sends the first $subarchive$ to explore. Upon receiving this message, an agent first forwards it to all its children before computing its local neighborhood (line 17). If the agent is a leaf in the pseudo-tree, it can directly send this neighborhood to his parent. The $neighbors$ message (line 22) is used to bring the neighborhood of each agent back to the root. An agent receives this message from its

children, combining the received neighborhood with its local one (line 23). Once the neighborhood of each children have been received, the resulting set of non-dominated solutions are sent to the parent node (line 26). In the case of the root agent, we now have the complete set of non-dominated solutions of the current $subarchive$. We consider assignments in the $subarchive$ to now be explored (line 28), and the newly found solutions are added to $\mathcal{A}$ (line 29). A new $archive$ (line 30) and $subarchive$ (line 35) are generated, and a new iteration is started by sending the $subarchive$ to the children. If at this point the $subarchive$ is empty, the search terminates and $\mathcal{A}$ contains the non-dominated solutions encountered during the search. To end the algorithm, the root sends $terminate$ messages (line 11) that are propagated by every agent down the pseudo-tree (line 13).

3.2.1.2 Properties

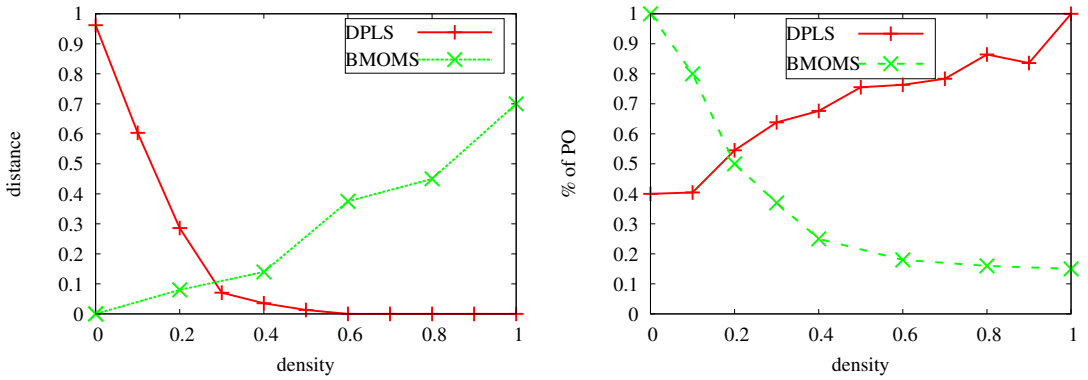
Property 5 (Termination). DPLS terminates when all encountered non-dominated solutions ($\mathcal{A}$) have been explored, i.e., we searched their neighborhood for new solutions. This means that in the worst case, DPLS terminates once the whole search space has been explored.

Proof. DPLS terminates once the $archive$, the set of assignments in $\mathcal{A}$ that are not yet explored, becomes empty. Since a problem has a finite number of possible assignments and at each iteration some assignments in $\mathcal{A}$ become explored, the $archive$ eventually becomes empty and DPLS terminates. □

Property 6 (Anytime). DPLS is anytime, i.e., at the end of each iteration the root knows all the non-dominated solutions encountered so far (maintained in $\mathcal{A}$).

Proof. At each iteration, we explore the neighborhoods $\mathcal{N}(A), \forall A \in subarchive$. After that, non-dominated solutions within the neighborhoods are added to $\mathcal{A}$ and all assignments in the $subarchive$ are considered $explored$. Since a solution in $\mathcal{A}$ is discarded only if it is dominated by another solution, we can guarantee the following at the end of each iteration: $\mathcal{A} = ND(\bigcup_{A|explored(A)} \mathcal{N}(A))$. □

Property 7 (Size of Neighborhood Messages). The maximum message size that an agent can send to its parent depends on its number of descendants (d), the maximum number of assignments that can be considered neighbor of some assignment A ($|\mathcal{N}(A)|$), and the number of assignment we explore at once (e). In the worst case, all solutions of a neighborhood $\mathcal{N}_i(A)$ are non-dominated, $\forall i \forall A$. The local non-dominated neighborhood of an agent will thus always be of size $|\mathcal{N}_i(A)| \times e$, assuming no overlapping neighborhood. Since neighborhood are accumulated when sent up the tree, an agent will receive $d \times |\mathcal{N}_i(A)| \times e$ solutions for its children and will send $(d+1) \times |\mathcal{N}_i(A)| \times e$ solutions to its parent.



(a) Min. distance from a Pareto optimal solution

(b) Fraction of PO found

FIGURE 3.5: Results for DPLS and B-MOMS on multi-objective graph-coloring instances with $|V| = 14$.

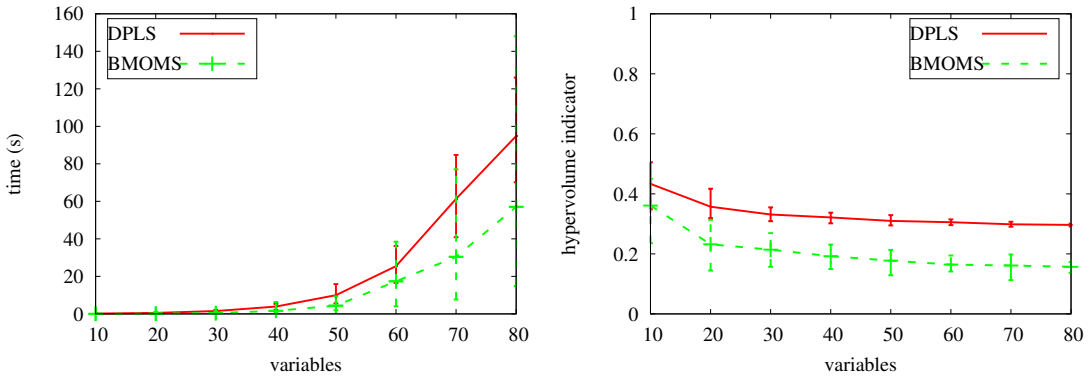
To simplify the algorithm, we do not detail how each agent can determine whether the cost of a local neighbor solution is non-dominated or not. In practice, each solution A in $subarchive$ is associated with its cost vector $R(A)$, allowing each agent to compute the cost of any of its neighbor by adding the impact of the local changes made to A to its original cost $R(A)$. The advantage of this approach is that each solution in $neighbors$ messages can be checked for dominance with regards to the received $subarchive$. However, it adds to the size of the $subarchive$ message, but compared to the size of the $neighbors$ messages, we consider it negligible. The main problem it poses is that it limits the neighborhood operators we can use. Since every agent must be able to compute the impact of its local changes, it must know the cost functions involving all the variables it changes, limiting us to considering operators that change the local variable of the agent and the variables it shares a constraint with.

3.2.2 Experimental Evaluation

In this subsection, we evaluate the performances of DPLS for various settings and compare it with B-MOMS [12], the state-of-the-art approximation algorithm for MO-DCOP.

3.2.2.1 Experimental Setting

We perform experiments using randomly generated multi-objective graph-coloring instances (see section 2.5.2). To produce our instances, random connected graphs were generated by specifying $|V|$ the number of variables and δ the density of the graph. When $\delta = 0$, the number of edges in the graph is equal to $|V| - 1$ and when $\delta = 1$, the number of edges is equal to $\frac{|V|}{2}(|V| - 1)$.



(a) Runtime (b) Hypervolume

FIGURE 3.6: Results with $\delta = 0.5$ and varying $|V|$.

In order to evaluate the performances of DPLS and compare them with B-MOMS, we use three different metrics, namely *minimum Euclidean distance*, *fraction of Pareto optimal solutions found* and the *hypervolume indicator*. Let PO be a set of all Pareto optimal solutions of an MO-DCOP and $\widetilde{PO}$ be an approximation of PO obtained by DPLS and B-MOMS. The *minimum Euclidean distance* measure the minimum distance between solutions $A \in \widetilde{PO}$ and an optimal solution $A^o \in PO$. The *fraction of Pareto optimal solutions found* is expressed as: $\frac{|\widetilde{PO} \cap PO|}{|PO|}$. The *hypervolume indicator* [66] is a measure of the volume of the objective space covered by a set of solutions. This last measure has the advantage of not requiring the exact set of Pareto optimal solutions, requiring instead a reference point to define the range of the objective space. For the multi-objective graph-coloring problem, the cost of each objective is bounded by the number of constraints in the problem and we use these bounds as reference point.

To compute PO, we use a brute-force optimal algorithm whose complexity is exponential in the number of variables ($|V|$). We thus could not compute PO for large instances and only report the *distance* and *fraction of Pareto optimal solutions* for problem instances with $|V| = 14$. For $|V| > 14$, we use the *hypervolume indicator* to measure the ratio of optimal solution space covered by $\widetilde{PO}$. We consider as optimal solution space the hypervolume ranging from the reference point to the utopia point where all objectives are equal to 0.

For all experiments, DPLS was started with 100 randomly generated assignments and each iteration explored the neighborhood of one assignment ($e = 1$).

3.2.2.2 Experimental Results

Figure 3.5 shows the results obtained with B-MOMS and DPLS over 40 multi-objective graph coloring instances with $|V| = 14$ and varying the density. Figure 3.5(a) shows the minimum Euclidean distance between $\widetilde{PO}$ and PO, and Figure 3.5(b) shows the ratio of Pareto optimal

solutions found by B-MOMS and DPLS. At low densities, DPLS easily fall into a local optima where changing the value of a single variable does not improve the solution cost. This causes the quality of DPLS to vary a lot depending on the instances, sometimes finding the full Pareto front and other times not finding a single Pareto optimal solution. This leads, for density $\delta < 0.2$, to finding an average of only around 40% of all Pareto optimal solutions. This causes DPLS to find only around 40% of all Pareto optimal solutions for density $\delta < 0.2$. B-MOMS, which is exact for $\delta = 0$, can still provide 80% of the Pareto solutions for $\delta = 0.1$. When density increases and each variable is involved in more constraints, the chances for improvements using local search also increases and DPLS finds more than 60% of all optimal solutions when density is above 0.3. B-MOMS, which relies on removing edges from the constraint hypergraph, suffers a great decline in solution quality when the density increases, finding less than 25% of PO for $\delta > 0.4$.

We now consider results obtained when varying $|V|$ between 10 and 80 with density $\delta = 0.5$. Figure 3.6 shows the average runtime and hypervolume indicator obtained over 40 instances. We first notice that the runtime of both B-MOMS and DPLS increases exponentially with the number of variables, following the increase in the size of the search space. Regarding the quality of the solutions obtained, we see that for quite dense graphs, DPLS always finds better approximations, with the gap between the two algorithms increasing from a 5% difference for $|V| = 10$ to a 20% difference for $|V| = 80$.

In summary, we showed that DPLS is able to find significantly better solutions than B-MOMS on problem of medium and high density, finding more optimal solutions for density $\delta > 0.25$. For reference, graph-coloring instances based on geographical constraints can have a density of up to $\delta = 0.63$. While DPLS is slightly slower than B-MOMS, both algorithms share a similar time complexity that is exponential in the number of variables. DPLS however has the advantage of being anytime.

3.2.3 Conclusion

In MO-DCOPs, since finding all Pareto optimal solutions is not realistic, it is important to consider fast but approximation algorithms. In this section, we developed a novel approximation algorithm called Distributed Pareto Local Search (DPLS) algorithm. DPLS uses a local search approach to generate an approximation of the Pareto front of an MO-DCOP. In the experiments, we evaluated the performance of DPLS on randomly generated multi-objective graph coloring instances and compared the obtained results with B-MOMS, the state-of-the-art approximation algorithm for MO-DCOPs. Our experiments showed that DPLS finds better approximations than B-MOMS in problems with medium or high density.

As future works, we plan on further studying several aspects of the DPLS. First, we want to experiment with the anytime property of DPLS and study ways to improve this property, by

using specific exploration strategies for example. This ability to stop the algorithm at any time with the assurance to have the best solutions encountered so far is very important when the available processing time is not known in advance. Another aspect we want to study is how to adapt some neighborhood operators to a distributed environment. Some existing operators are very successful on specific problems and could be interesting to use for the DPLS. However, these operators can involve several variables and using them in distributed settings is not trivial as they can produce overlapping local neighborhoods, requiring additional coordination between agents. Similarly, exploration and restart strategies should be implemented in distributed settings as they proved to be quite successful for the centralized PLS [67]. Another method that should be studied for DPLS is the Queued PLS [68] that delays the removal of dominated solution from the archive as it can improve the approximation found by the algorithm. Furthermore, we intend to apply DPLS to challenging real world problems, e.g., sensor network and scheduling problems.

Chapter 4

Solution Selection

In this chapter, we present methods for selecting a solution or a set of solutions from a Pareto front. We showed in the previous chapter that the first step of solving multi-objective problems consists in finding the Pareto front of the problem. Once a set of solution is found, a single solution should be extracted in order to implement it in reality. The selection of this solution can be performed in several ways, either by involving a decision maker or not. If a human is asked to pick the solution, presenting him with the full Pareto front might be overwhelming, especially if the Pareto front contains a large number of solutions with many objectives. In that case, even if we let a human make the final decision, we can help this decision process by presenting him with only an interesting subset of the Pareto front. If no decision maker is able to make the selection or if there is not time to wait for its decision, then a fully automated selection method should be used.

In section 4.1, we present a selection method based on the weighted-sum, a scalarization techniques which, given some weights, aggregates the different objectives into a single value. In section 4.2, we present a selection method for dynamic problems based on the concept of resilience. In section 4.3, we present a selection method based on limiting the implementation cost of the selected solution.

The work of section 4.1 has been presented at the 11th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2016) [52]. The work of section 4.2 has been presented at the International Conference on Agents and Artificial Intelligence (ICAART 2015) [53].

4.1 Sum-Based Selections Applied to Multi-Objective Timetabling Problems

The *university course timetabling* problem [69–71] is one of the representative application problems in operations research (OR) and artificial intelligence (AI) which can be generally defined as the task of assigning a number of lectures to a limited set of time slots and rooms, subject to a given set of hard and soft constraints. This problem is formalized as a combinatorial constraint optimization problem where the aim is to find an assignment of values to variables so that all hard-constraints are satisfied and the sum of all violated costs of soft-constraints is minimized.

Many real-world university course timetabling problems involve multiple criteria that can be considered separately but should be optimized simultaneously. In a multi-objective course timetabling, there usually exists an exponential number of acceptable solutions called Pareto-optimal solutions. An assignment is considered Pareto-optimal if there does not exist another assignment that is better for all objectives. Complete approaches computing the full set of Pareto-optimal solutions do not scale well for complex problems such as timetabling. A common compromise is to use a scalarization method, which is the simplest and the most widely used method to find Pareto-optimal solutions in a multi-objective timetabling problem. Such scalarization methods effectively transform a problem with multiple objectives into a mono-objective problem. The optimal solution obtained is then guaranteed to be Pareto-optimal and to satisfy an utilitarian view where the sum of objectives is optimized. Various sophisticated approximation algorithms have also been developed for finding feasible solutions, e.g., simulated annealing [72], multi-phase tabu search [71] and multi-objective evolutionary algorithm [73]. However, they cannot guarantee to find Pareto-optimal solutions.

The focus of our work is laid on finding a variety of Pareto-optimal solutions to the university course timetabling problem by using scalarization methods. Timetabling is a highly symmetrical problem, where the neighborhood of a valid timetable will often contain a number of other valid timetables. Because of this symmetry, there often exists a huge number of solution to a timetabling problem, several of which can optimize the weighted-sum. Assume that there exists three criteria/objectives, e.g., room capacity, minimum working days and isolated lectures, and the aim is to minimize the sum of all violated costs of these three objectives. Let 8 be the minimal value obtained by using the scalarization method, e.g., 5 for the violated cost of room capacity and 3 for that of isolated lectures. Now, what happens when there exists several solutions that have the same minimal value, e.g., 2 for first criterion, 5 for the second and 1 for the last. The existing scalarization method cannot capture all these solutions, since it terminates when one optimal solution has been found. However, from multi-objective perspective, these solutions can be very different from each other, offering various trade-offs of objectives and a partial

representation of the Pareto front. As far as we are aware, there is no other work that focuses on providing a set of optimal solutions for university course timetabling.

In this paper, we propose a new approach for multi-objective timetabling problems. By treating the violations of each soft-constraint as an objective, we can associate a vector of penalties to any timetable. Then, we propose to compute the set of all utilitarian vectors, offering different trade-offs of objectives while minimizing their sum. We call this set the $\sum_x$-optimal front. Solutions in this set are guaranteed to be Pareto-optimal as well as satisfy utilitarianism, i.e., the sum of the violated costs is minimal. However, as we will show in this paper, the $\sum_x$-optimal front can be quite large on some problem instances. Because of the complexity of university timetables, directly providing too many solutions to the decision makers can be of little interest. Therefore we also introduce additional criteria to extract specific solutions from the $\sum_x$-optimal front in order to provide an interesting set of solutions.

In the experiments, we use Answer Set Programming (ASP) [74–77] to compute the complete $\sum_x$-optimal front of some popular university course timetabling instances used in the ITC-2007 competition [46]. ASP is an approach to declarative problem solving, combining a rich yet simple modeling language with high-performance solving capacities. It is well suited for implementing our approach as it was already used to compute some optimal solutions for CB-CTT in a previous work [48]. Using ASP, we are able to compute solutions of many of the benchmark instances and we show that many trade-offs are indeed available, validating the strength of our approach.

The rest of the paper is organized as follows. In 4.1.1, we define multi-objective concepts for CB-CTT and introduce a novel solution criterion called $\sum_x$-optimality. In 4.1.2, we discuss additional criteria we can use to isolate interesting solutions from the $\sum_x$-optimal front. In 4.1.3, we experiment our novel approach with university course timetabling benchmarks from ITC-2007. We then conclude in 4.1.4.

4.1.1 Set of Solutions

In the previous subsection, we showed how to evaluate the quality of a timetable when preferences over the soft-constraints are known. Each timetable can be associated with a single value and most works on timetabling will find one timetable minimizing this value.

There are two issues with this approach, (i) it only provides the decision makers with one possibility and (ii) it overlooks the various trade-offs of penalties available. In timetabling and many other decision problems, two solutions that might appear equivalent for a given criteria (same weighted-sum) might actually be very different in the eyes of the decision makers. For example, there might exists three vectors $\{9, 1\}$, $\{5, 5\}$ and $\{1, 9\}$ that share the same sum (10) but with very different trade-offs of objectives.

In this subsection, we will consider how to provide more than one solution to the Curriculum-Based Course Timetabling (CB-CTT) problem by improving upon the weighted-sum criterion.

4.1.1.1 Pareto Front

For a given CB-CTT, each timetable can be associated with a weighted-vector of penalties. By considering each penalty as an objective to optimize, we can introduce the concept of Pareto-optimality for timetables.

Definition 4.1 (Pareto Dominance)**.** Given two vectors V and V', we say that V *Pareto dominates* V', denoted by $V \succ V'$, iff V is partially less than V', i.e., (i) it holds $v_i \in V \leq v'_i \in V'$ for all i, and (ii) there exists at least one i such that $v_i \in V < v'_i \in V'$

Definition 4.2 (Pareto-Optimal Solution)**.** A timetable T is Pareto-optimal for UD_x if there does not exist another timetable T' such that $V_x(T) \prec V_x(T')$.

The penalty vectors corresponding to those Pareto-optimal timetables form what is called the Pareto Front.

Definition 4.3 (Pareto Front)**.** The Pareto Front is the set of vectors we can obtain from all Pareto-optimal solutions.

By finding one solution for each possible vector of the Pareto front, we can provide every interesting trade-off of penalties. However, finding the Pareto front of complex problems is often too difficult. In the worst case, each assignment can be Pareto-optimal and with large problems such as CB-CTT, we can have thousands of variables to assign to thousands of pair room/period. The number of solutions being exponential, using complete methods becomes unrealistic on large problem instances. Because of this complexity, it is important to consider techniques that can

TABLE 4.1: Example of Solutions

	Penalties			
	S_1	S_2	S_3	S_5
T_1	5	0	3	2
T_2	6	1	0	3
T_3	7	1	1	5
T_4	2	1	2	2
T_5	5	0	3	5

quickly provide an approximation of the Pareto front. We thus propose to find a subset of the Pareto front while still complying with the weighted-sum criterion.

4.1.1.2 $\sum_x$-Optimal Front

Since the weighted-sum is the state of the art approach for CB-CTT, we can assume that a timetable that is $\sum_x$-optimal is a good solution. However, there might exists more than one such solution, potentially offering very different trade-offs between penalties. Finding the set of all $\sum_x$-optimal solutions would allow to offer a set of good timetables to choose from. The user can then select one of the possible choices by considering his preferred trade-off of objectives, or even criteria that were not originally formulated in the problem. Since those solutions minimize the weighted-sum of the penalties, they are all Pareto-optimal, meaning that there exists no other timetable strictly better for all objectives.

Property 8 (Pareto-Optimality of $\sum_x$-Optimal Solutions). A vector V that is $\sum_x$-optimal is also Pareto-optimal for UD_x.

Due to the size and high symmetry of CB-CTT problems, there might exists a huge number of timetables that share the same penalty vector. In our work, we are focusing on providing various penalty trade-offs and do not care about providing more than one timetable for each possible vector. Thus, we will focus on the subset of the Pareto front we can find using $\sum_x$-optimal solutions.

Definition 4.4 ($\sum_x$-Optimal Front)**.** The $\sum_x$-optimal front is the set of vectors that can be obtained by $\sum_x$-optimal solutions.

4.1.1.3 Example

Let us now review all the concepts introduced in this subsection using a simple example.

Example 4.1 (CB-CTT)**.** *Let us consider an example of CB-CTT that accepts five valid timetables T_1, T_2, T_3, T_4 and T_5 whose penalty vectors are shown in Table 4.1, where column S_i contains the penalty for soft-constraint S_i and line T_i contains the vector $V(T_i)$.*

First, let us consider formulation UD_2. We consider the four objectives S_1, S_2, S_3 and S_5 with weights 1, 5, 2 *and* 1 *respectively. By applying the set of weight to each vector, we obtain the following weighted vectors:* $V_2(T_1) = \{5, 0, 6, 2\}$, $V_2(T_2) = \{6, 5, 0, 3\}$, $V_2(T_3) = \{7, 5, 2, 5\}$, $V_2(T_4) = \{2, 5, 4, 2\}$ *and* $V_2(T_5) = \{5, 0, 6, 5\}$.

Based on the definition of Pareto-optimality, this problem has three Pareto-optimal timetables when using UD_2: T_1, T_2 and T_4. T_3 is not Pareto-optimal since it is dominated by T_2 ($\{6, 5, 0, 3\} \succ \{7, 5, 2, 5\}$) and T_5 is dominated by T_1 ($\{5, 0, 6, 2\} \succ \{5, 0, 6, 5\}$).

The Pareto front for this problem is the set of vectors obtained from Pareto-optimal solutions: $\{\{5, 0, 6, 2\}, \{6, 5, 0, 3\}, \{2, 5, 4, 2\}\}$.

Now if we consider the weighted sum of each vector, we can find two $\sum_2$-optimal solutions, T_1 and T_4. Those timetables minimize the sum of weighted penalties with a sum equal to 13 ($5 + 0 + 6 + 2 = 13$ for T_1 and $2 + 5 + 4 + 2 = 13$ for T_4). T_2 is not $\sum_2$-optimal since its corresponding sum is not minimal($6 + 5 + 0 + 3 = 14$).

We can then say that the $\sum_2$-optimal front of this problem is $\{\{5, 0, 6, 2\}, \{2, 5, 4, 2\}\}$.

Let us now consider formulation UD_1. This time we only consider three objectives S_1, S_2, S_3 with weights 1, 5, 1 *respectively. We obtain the following weighted vectors:* $V_1(T_1) = \{5, 0, 3\}$, $V_1(T_2) = \{6, 5, 0\}$, $V_1(T_3) = \{7, 5, 1\}$, $V_1(T_4) = \{2, 5, 2\}$ *and* $V_1(T_5) = \{5, 0, 3\}$.

We now have four Pareto-optimal timetables: T_1, T_2, T_4 and T_5.Only T_3 is not Pareto-optimal since it is still dominated by T_2 ($\{6, 5, 0\} \succ \{7, 5, 1\}$).

The Pareto front for this problem is the set of vectors obtained from Pareto-optimal solutions: $\{\{5, 0, 3\}, \{6, 5, 0\}, \{2, 5, 2\}\}$.

Now if we consider the weighted sum of each vector, we can find two $\sum_1$-optimal solutions, T_1 and T_5. Those two timetables minimize the sum of weighted penalties with a sum equal to 8 ($5 + 0 + 3 = 8$). T_2 and T_4 are not $\sum_1$-optimal since their corresponding sum are not minimal ($6 + 5 + 0 = 11$ for T_2 and $2 + 5 + 2 = 9$ for T_4).

When we consider the $\sum_1$-optimal front, we only care about the vectors and not the assignment (timetable). Since here both T_1 and T_5 share the same vector, we have the $\sum_1$-optimal front $\{\{5, 0, 3\}\}$.

Next, we will consider additional criteria that we can use to identify solutions from the $\sum_x$-optimal front with interesting properties.

4.1.2 Subset of the $\sum_x$-Optimal Front

With the $\sum_x$-optimal front, we can provide a set of solutions to a decision maker. While we can expect this set to be much smaller than the Pareto front, its size might still prove too large to handle for a human. A typical university timetable being made of thousands of lectures assigned to thousands of rooms and periods, if a human wants a good understanding of the different timetables, it is imperative to present him with a limited set of solutions. In the case of multiple decision makers, proposing a large number of alternatives can also results in each decision maker preferring a different timetable. Thus we believe it is important to consider additional criteria when proposing a set of solutions.

The solutions provided by a $\sum_x$-optimal front already satisfy an utilitarian criterion [56], guaranteeing a minimal sum of penalties. We will now propose two additional criteria we can consider to isolate specific solutions from the $\sum_x$-optimal front.

4.1.2.1 Utilitarianism

The weighted-sum is a well known utilitarian operator [56]. This means that solutions from the $\sum_x$-optimal front are guaranteed to provide timetables with the least amount of (weighted) penalties. Since in CB-CTT each penalty corresponds to a violated constraint in the timetable, this is a very desirable property. The downside of utilitarianism is that it completely ignores any kind of balance between the different objectives. For example, this means we can obtain timetables where all the penalties come from overcrowded rooms without violating any other soft-constraint. The opposite situation would be to have the same amount of (weighted) penalties for each soft-constraint.

By providing the complete $\sum_x$-optimal front, we can provide all those variations if they exist.

4.1.2.2 Egalitarianism

With egalitarianism, instead of focusing exclusively on the weighted-sum, the goal is to obtain a balanced solution where penalties are spread as much as possible between the objectives (soft-constraints). We propose here to extract the most egalitarian solution from the $\sum_x$-optimal front. Since we already know those solutions satisfies utilitarianism, we can obtain the most well balanced solutions that also minimizes the weighted-sum. To find the most egalitarian solutions from the $\sum_x$-optimal front, we use a lexicographic ordering [56]. Basically, we want to find the largest minimum value. This requires to arrange a vector V in an increasing order, denoted $V^<$. We then define the following ordering:

Definition 4.5 (Lexicographic ordering)**.** Given two vectors V and V' of size m, we say that V *lexicography precedes* V', denoted by $V \prec^{lex} V'$, iff there is a $i \in \{1, \ldots, m\}$ such that $V_i^< < V_i'^<$ and, if $i > 1$, then $\forall j \in \{1, \ldots, i-1\}, V_j^< = V_j'^<$.

The most egalitarian vector from the $\sum_x$-optimal front is the one that is preceded by all others in the lexicographic ordering, which tends to maximize the minimum value of its vector. With CB-CTT, this solution will tend to have the most balanced penalties, which can be interesting if many small constraint violations are considered better than a few constraints being violated numerous times. For other models of the timetabling problems, egalitarianism can be used to find fair solutions [78], for example to have penalties spread between different school departments, different teachers, different classes, ...

Example 4.2 (Egalitarianism)**.** *Let us consider the same example as Example 1 with the problem shown in Table 4.1. Without applying any weight, we can order the five vectors of penalty using the lexicographic ordering. First, let us reorder each vector* $V(T_i)$ *into* $V^<(T_i)$*. We obtain* $V^<(T_1) = \{0, 2, 3, 5\}$*,* $V^<(T_2) = \{0, 1, 3, 6\}$*,* $V^<(T_3) = \{1, 1, 5, 7\}$*,* $V^<(T_4) = \{1, 2, 2, 2\}$ *and* $V^<(T_5) = \{0, 3, 5, 5\}$*. Here,* $V(T_2)$ *lexicography precedes all other vectors. Its first value (corresponding to the minimum) in* $V^<(T_2)$ *is* 0*, which is inferior to the first value in* $V^<(T_3)$ *and* $V^<(T_4)$ *(*$0 < 1$*), thus* $V(T_2) \prec^{lex} V(T_3)$ *and* $V(T_2) \prec^{lex} V(T_4)$*. Its first value is equal to the first value in* $V^<(T_1)$ *and* $V^<(T_5)$*, but then the next value (*1*) is inferior (*2 *for* T_1 *and* 3 *for* T_5*), thus we have* $V(T_2) \prec^{lex} V(T_1)$ *and* $V(T_2) \prec^{lex} V(T_5)$*. We can say that* T_2 *is the less balanced of the solutions.*

The most balanced solution, said to be egalitarian, is T_4 *as it is lexicography preceded by all the other vectors. Its minimum value of* 1 *is larger than the minimum value of* T_1*,* T_2 *and* T_5 *and we can write* $V(T_1) \prec^{lex} V(T_4)$*,* $V(T_2) \prec^{lex} V(T_4)$ *and* $V(T_5) \prec^{lex} V(T_4)$*. Its minimum value of* 1 *is the same as* T_3*, but then the next value (*2*) is larger than in* T_3 *(*1*), thus* $V(T_3) \prec^{lex} V(T_4)$*. We can say that* T_4 *is the most balanced vector and is the egalitarian solution here.*

4.1.2.3 Constraint Satisfaction

Another aspect we can consider is the maximization of the number of completely satisfied soft-constraints. No violation of some objectives can be a good selling point for a decision maker and it is easy to explain the different with another timetable that does not satisfy the same constraints.

This focus on satisfying as many soft-constraints as possible could be simply expressed in the original problem. However, if we focus on the satisfaction of the objectives, instead of their optimization, we can end up with bad solutions that have a huge amount of violations on a few objectives.

Here, since we work from the $\sum_x$-optimal front, we always have a minimal sum of weighted penalties. Such solutions can be interesting if dealing with a few big problems is more desirable. In a model where each objective represents a different agent, such criteria should not be satisfied but can be used to remove solutions that we know are not fair.

In the next subsection, we will present experimental results and show the interest of those criteria.

4.1.3 Experiments

In this subsection, we first present our method to enumerate all vectors in the $\sum_x$-optimal front before showing some results obtained on instances from the ITC-2007 competition [46].

4.1.3.1 Method

To conduct those experiments, we used an encoding of the CB-CTT with Answer set programming (ASP) that was shown to be a promising approach for timetabling [48], offering an efficient way to find one $\sum_x$-optimal solution of a problem.

ASP is a form of declarative programming mostly used to solve NP-hard problems. An ASP problem is made up of rules that represents conditions to be satisfied and facts that are known to be true. An ASP solver will encode such problem into a logic program before searching for some stable models, corresponding to solutions of the original problem.

We represent the CB-CTT problem as a set of rules and a problem instance is a set a set of fact such as the number of days, period per days, courses, teachers, ... We can then search for a timetable that satisfies all the rules from the hard-constraints and that minimizes the sum of penalties from the soft-constraints.

While in theory it is possible to enumerate all valid timetables, in practice, due to the symmetry of CB-CTT, it is too complex and the enumeration takes too much time. Thus we focus on the vector of penalties and only find one corresponding timetable.

In the method we used, we find $\sum_x$-optimal solutions one by one. For each newly found vector of penalty, we extend the original problem by adding a rule forbidding the same vector to be a solution. This method ends when there exists no more solution to our extended problem, meaning that we found the complete $\sum_x$-optimal front.

Because the time to find a new solution can be very long, we can note that it is possible to stop at anytime during our method and then use the $\sum_x$-optimal solutions found so far. In the best case, we stopped during the last step which is only used to prove that we indeed found the complete

TABLE 4.2: $\sum_4$-optimal front for comp17 (with optimal weighted sum of 21)

#	S_1	S_2	S_4	S_6	S_9	#	S_1	S_2	S_4	S_6	S_9
1	0	13	0	8	0	12	0	12	0	9	0
2	0	11	3	7	0	13	0	11	0	10	0
3	0	12	2	7	0	14	0	12	3	6	0
4	0	11	1	9	0	15	0	9	4	8	0
5	0	12	1	8	0	16	0	10	4	7	0
6	0	11	2	8	0	17	0	11	4	6	0
7	0	10	3	8	0	18	0	14	1	6	0
8	0	10	2	9	0	19	0	14	0	7	0
9	0	9	3	9	0	20	0	15	0	6	0
10	0	13	2	6	0	21	0	10	1	10	0
11	0	13	1	7	0						

$\sum_x$-optimal front. Else, we only have an incomplete $\sum_x$-optimal front that can still be used and might still provide a number of interesting trade-offs.

4.1.3.2 Results

We now present results obtained by using our method on some instances proposed for the International Timetabling Competition of 2007 [46]. There is a total of 21 instances of various complexity. The solving of some of those instances are still open with any $\sum_x$-optimal solutions yet to be found.

Table 4.2 shows the $\sum_4$-optimal solutions of instance *comp17*. For the formulation UD_4, *comp17* is a difficult instance that had no known $\sum_4$-optimal solutions until now. As we can see, this instance has 21 $\sum_4$-optimal solutions, minimizing the sum of penalties at 21. While soft-constraints S_1 and S_9 are always fully satisfied, we can observe different trade-offs over the 3 other objectives. S_2 varies between 9 and 15, S_4 varies between 0 and 4, and S_6 varies between 6 and 10. Those different trade-offs can be very interesting to a decision maker and we can notice some clear differences between the available vectors. Most notably, some vectors can completely satisfy the constraint S_4. As we discussed in the previous subsection, it can sometimes be preferable to completely satisfy one more objective at the cost of increasing the penalty of two others (vector #1 with penalties $\{0, 13, 0, 8\}$ for example). We also point out the vector #15 with penalties $\{0, 9, 4, 8, 0\}$, which is the most egalitarian vector, offering a good balance between S_2, S_4 and S_6.

Table 4.3 shows the $\sum_4$-optimal solutions of instance *comp04*. As we can see, this instance has 13 $\sum_4$-optimal solutions, minimizing the sum of penalties at 13. Like in *comp17*, soft-constraints S_1 and S_9 are always fully satisfied. However, the variations of the three others objectives are quite different, especially S_4 which only varies between a penalty of 0 and 1. We can thus notice that for each vector where $S_4 = 1$, we can find a very close vector where S_4 is

TABLE 4.3: $\sum_4$-optimal front for comp04 (with optimal weighted sum of 13)

#	S_1	S_2	S_4	S_6	S_9	#	S_1	S_2	S_4	S_6	S_9
1	0	10	0	3	0	8	0	8	1	4	0
2	0	9	1	3	0	9	0	11	0	2	0
3	0	8	0	5	0	10	0	6	0	7	0
4	0	9	0	4	0	11	0	12	0	1	0
5	0	7	0	6	0	12	0	11	1	1	0
6	0	10	1	2	0	13	0	6	1	6	0
7	0	7	1	5	0						

completely satisfied (vector #2 $\{0, 9, 1, 3, 0\}$ and #1 $\{0, 10, 0, 3, 0\}$ for example). If we value the complete satisfaction of the objectives, we can consider ignoring vectors that violates S_4 but are very close to vectors where $S_4 = 0$. However, if we care more about having a good balance between the penalties, the best solution here would be the vector #13 $\{0, 6, 1, 6, 0\}$. This vector, while providing the most balance, still greatly favors S_2 and S_6 compared to S_4, showing that we cannot always find a perfectly-balanced solution.

Previous approaches that focus on finding one solution usually produce only one of the vectors we show here. In light of the many possible trade-offs and the clear differences between two $\sum_x$-optimal solutions, we showed the importance to present those alternatives to a decision maker.

Table 4.4 shows the runtime, the size of the $\sum_x$-optimal front as well as the optimal sum of penalties for some instances and formulations. Due to the very long time it takes to compute the $\sum_x$-optimal front, we have yet to produce results for all possible instances and formulations and for this experiment, we limited the time to find a new solution (or prove unsatisfiability) to 6 hours. Each line in the table represent data for a given instance and formulation. The size represent the size of the $\sum_x$-optimal fronts and sum represents the sum of penalties for the vectors found. Total time represents the total CPU time it took to find the complete $\sum_x$-optimal front. Unsat time is the duration of the last step of our method, which requires to prove that with the added constraints, the problem is no longer satisfiable, meaning that we found the complete $\sum_x$-optimal front.

While we were able to solve many instances using our method, we were only able to completely find $\sum_x$-optimal front for the formulation UD_4. Because UD_4 uses an additional hard-constraint, the number of valid timetables is greatly reduced compared to other formulations, making it possible to find a new $\sum_4$-optimal solution within the 6 hours limit.

For other formulations, we are only able to solve instances with a unique $\sum_x$-optimal solution.

Regarding the runtime, it will greatly vary based on the complexity of the instance and the formulation used. Additionally, since we solve the same problem several times, adding a constraint after each newly found vector, we can expect that the higher the size of the $\sum_x$-optimal front,

TABLE 4.4: Results for finding the $\sum_x$-optimal front on various instances

instance	formulation	size	total time (s)	unsat time (s)	sum
comp04	UD3	1	1.8	0.92	2
comp04	UD4	13	764.02	593.66	13
comp04	UD5	1	12705.82	12664.52	49
comp06	UD2	1	1290.64	1102.6	27
comp06	UD3	1	7.26	3.78	8
comp07	UD1	1	48.22	9.72	3
comp07	UD2	1	18206.52	11129.4	6
comp07	UD3	1	214.38	0.98	0
comp07	UD4	4	19966.38	241.94	3
comp08	UD3	1	16.18	7.12	2
comp08	UD4	14	2316.62	1343.14	15
comp08	UD5	1	2610.72	2524.4	55
comp10	UD1	1	2.8	1.42	2
comp10	UD2	1	1565.24	925.84	4
comp10	UD3	1	2.96	0.8	0
comp10	UD4	6	106.48	12.4	3
comp11	UD1	1	0.28	0.12	0
comp11	UD2	1	0.4	0.14	0
comp11	UD3	1	0.72	0.32	0
comp11	UD4	1	3.38	1.08	0
comp11	UD5	1	838.66	0.58	0
comp14	UD3	1	1.86	0.58	0
comp14	UD4	2	55.04	31.4	14
comp16	UD1	1	4207.06	4203.16	11
comp16	UD2	1	48.04	13.8	18
comp16	UD3	1	4.18	2.56	4
comp16	UD4	3	127.5	60.96	7
comp17	UD3	1	6.8	3.78	12
comp17	UD4	21	341,681.21	42,963.86	21
comp18	UD3	1	1.14	0.44	0
comp20	UD1	1	767.78	389.74	2
comp20	UD2	1	763.46	210.4	4
comp20	UD3	1	371.94	0.92	0

the longer the time it takes to completely find it. It results that some instance were solved in less than one second (*comp11* for UD_1, UD_2 and UD_3) while others took up to 30 minutes (*comp10* for UD_2) and even around 94 hours (*comp17* for UD_2 and UD_4). We can see from the unsat time column that for some instances, the majority of the time is taken to prove that there does not exist anymore vector belonging to the $\sum_x$-optimal front (*comp04* for UD_5 for example). While having the complete $\sum_x$-optimal front can be important, it can also be ignored if a decision needs to be taken within a short amount of time. Using our method as an anytime approach, we can provide a set of $\sum_x$-optimal solutions more quickly but at the cost of potential incompleteness.

TABLE 4.5: Results for finding the $\sum_x$-optimal front on various instances (using weights of 1)

instance	formulation	size	total time (s)	unsat time (s)	sum
comp04	UD1	8	84,216.68	83,929.64	12
comp04	UD2	8	71,468.52	71,434.84	12
comp04	UD3	1	2.04	1.00	1
comp04	UD5	1	25.64	13.28	13
comp07	UD1	4	168.46	30.52	3
comp10	UD1	2	3.38	1.41	2
comp10	UD2	2	1,883.54	552.79	2
comp10	UD3	1	2.94	0.82	0
comp11	UD1	1	0.28	0.12	0
comp11	UD2	1	0.86	0.14	0
comp11	UD3	1	0.98	0.32	0
comp11	UD5	1	1,121.50	0.58	0
comp16	UD1	1	5.24	2.82	5
comp16	UD2	1	33.56	7.66	5
comp16	UD3	1	13.18	11.32	2
comp17	UD2	5	337,888.42	336,166.34	19
comp17	UD3	2	42.72	14.69	6

TABLE 4.6: $\sum_2$-optimal front for comp04 (with weights of 1, optimal sum of 12)

#	S_1	S_2	S_3	S_5	#	S_1	S_2	S_3	S_5
1	0	10	2	0	5	0	7	5	0
2	0	9	3	0	6	0	12	0	0
3	0	11	1	0	7	0	6	6	0
4	0	8	4	0	8	0	5	7	0

4.1.3.3 Results with neutral weights

Choosing weights for each soft-constraint can be a difficult task for a decision maker, and the impact of those weights is hard to estimate in advance. Thus, it might sometimes be better to start by considering the neutral case where no preferences are given and where the decision maker simply needs to provide the constraints to take into account. We evaluated this case and ran experiments where we put the weights of the considered soft-constraints to 1, showing the results in Table 4.5. Because UD_4 already used weights of 1, we focus on the other formulations and are able to find interesting $\sum_x$-optimal fronts on a few instances.

For *comp04*, we were able to find $\sum_x$-optimal front of size 8 for both UD_1 and UD_2 (with weights of 1) and we show the corresponding $\sum_2$-optimal front in Table 4.6 In this case, S_1 and S_5 are both completely satisfied and the 8 solutions offer different trade-offs between S_2 and S_3. Only one vector (#6 with penalties $\{0, 12, 0, 0\}$) can completely satisfy three of the soft constraints and there exists one well balanced vector (#7 with penalties $\{0, 6, 6, 0\}$).

Those results show that it is possible to compute the $\sum_x$-optimal front of complex CB-CTT instances. We can also show that while some instances are highly symmetric, offering large $\sum_x$-optimal front, other instances only have one or very few vectors minimizing the sum. The different trade-offs offered seems interesting and could make university more confident when choosing a timetable from several alternatives.

4.1.4 Conclusion

In this section, we proposed a new approach to the Curriculum-Based Course Timetabling. Approaching the problem as a multi-objective one, we wanted to provide several timetables to choose from. Thus, we defined a subset of the Pareto Front we called $\sum_x$-optimal front. This subset contains all vectors that minimize a weighted-sum of penalties, the weighted-sum being the most used criteria for CB-CTT. In comparison, the majority of previous works using this criteria only provide one solution. We showed in our experiments however that two vectors minimizing a weighted-sum can be very different. Those differences are very important to a decision maker and should not be ignored. While this paper focuses on CB-CTT, we believe our approach can be applied to many other timetabling problems. Additionally, we proposed simple ways to isolate solutions from the $\sum_x$-optimal front. One interesting criteria is egalitarianism, where we search for a good balance between all objectives. Another consideration can be the satisfaction of as many objectives as possible. Usually, focusing on those criteria is made at the cost of the overall quality of the solution. Here, since we first focus on utilitarianism and then consider additional criteria, we always keep the guarantee of minimizing the sum of objectives.

In future works, we want to develop more efficient methods to compute a set of solutions for timetabling problems. We plan on using asprin [79], a tool that was recently proposed for expressing preferences in Answer Set Programming. Using this tool, we could directly specify the desired set of solutions (for example the $\sum_x$-optimal front) and compute it in one shot. In addition to an increased in efficiency, a number of new criteria could easily be implemented and will be the topic of future research.

We also plan on considering other subsets of the Pareto front that do not focus on the utilitarian criterion (minimizing a weighted-sum). It can be important in some cases to consider solutions where the sum of penalties is not minimal but where the trade-offs are more interesting. For example if the objectives represent penalties with different teachers, it can be more important to have the most egalitarian solution so that no teacher can feel jealous about other teachers. Another thing to consider is the automatic extraction of interesting solutions from the $\sum_x$-optimal front. Let us say we only want to end up with five solutions to present to the decision maker. We might decide we want one of those solutions to respect the egalitarian criteria. Then the other four should be automatically selected in order to offer a diverse set of possibilities. Additional

criteria Selecting a subset of the $\sum_x$-optimal front is a problem that goes beyond timetabling and is of great interest to the multi-objective community. Reference point Techniques from multi-objective optimization can be used, for example using a reference point [80] was successfully applied to timetabling problems.

Another approach that could be interesting to study is the search for a subset of representative solutions [62]. Since the Pareto front is often too hard to compute and too large to analyze by a human, it would interesting to be able to provide a subset that best represents the complete Pareto front. While the $\sum_x$-optimal front provide a subset of solutions with various trade-offs, more extremes solutions that are not $\sum_x$-optimal could also be included in order to represents all the alternatives available. Finally, we will also apply our approach to other timetabling or scheduling problems as they often involve multiple criteria.

4.2 Resilience-Based Solution Selection in Dynamic Multi-Objective DCOPs

Many researchers of different fields have recognized the importance of a new research discipline concerning the *resilience* of real world complex systems [81–84]. The concept of resilience has appeared in various disciplines, e.g., environmental science, materials science and sociology. The goal of resilience research is to provide a set of general principles for building resilient systems in various domains, such that the systems are resistant from large-scale perturbations caused by unexpected events and changes, and keep their functionality in the long run. Holling (1973) first introduced the concept of resilience as an important characteristic of a well-behaved ecological system, and defined it as the capacity of an ecosystem to respond to a perturbation or disturbance by resisting damage. He adopted a verbal, qualitative definition of ecological resilience, rather than a mathematical, quantitative one. "Resilience determines the persistence of relationships within a system and is a measure of the ability of these systems to absorb changes of state variables, driving variables, and parameters, and still persist." (Holling, 1973, page 17). Schwind et al.(2013) adopted a computational point of view of *Systems Resilience*, and modeled a resilient system as a dynamic constraint-based model (called SR-model), i.e., dynamic constraint optimization problem. They captured the notion of resilience using several factors, e.g., resistance, recoverability, functionality and stabilizability.

In order to capture and evaluate the resilience of realistic dynamic systems, it requires to (i) consider the several objectives simultaneously, i.e., dynamic constraint optimization problem with multiple criteria, and (ii) develop an algorithm for solving this problem.

In this section, Two solution criteria for solving DMO-COP are provided, namely *resistance* and *functionality*, which are properties of interest underlying the *resilience* for DMO-COPs. Our model is defined by a sequence of MO-COPs representing the changes within a system that is subject to external fluctuations. The resistance is the ability to maintain some underlining costs under a certain threshold, such that the system satisfies certain hard constraint and does not suffer from irreversible damages. The functionality is the ability to maintain a guaranteed global quality for the configuration trajectory in a sequence. These two properties are central in the characterization of *robust* solution trajectories, which keep a certain quality level and *absorb* external fluctuations without suffering degradation. Indeed, these notions are consistent with the initial formulation of resilience from [81].

An algorithm called *Algorithm for Systems Resilience* (ASR) for finding the resilient trajectory in a DMO-COP is presented. This algorithm is based on the branch and bound search, which is widely used for COP and MO-COP algorithms, and it finds all resistant and functional solutions for DMO-COPs. In the experiments, the performances of ASR are evaluated with different types of dynamical changes.

We believe that the computational design of resilient systems is a promising area of research, relevant for many applications like sensor networks. A sensor network is a resource allocation problem that can be formalized as a COP [85]. For example, consider a sensor network in a territory, where each sensor can sense a certain area in this territory. When we consider this problem with multiple criteria, e.g., data management, quality of observation data and electrical consumption, it can be formalized as an MO-COP [86]. Additionally, when we consider some accidents, e.g., sensing error, breakdown and electricity failure, it can be represented by the dynamical change of constraint costs.

The rest of the section is organized as follows. In 4.2.0.1, we define resistant and functional solutions in a DMO-COP. In 4.2.0.2, an algorithm called ASR is presented. In 4.2.1, some empirical results are provided. We finally conclude in 4.2.2.

TABLE 4.7: Cost table of MO-COP$_1$.

x_1	x_2	$cost$	x_2	x_3	$cost$	x_1	x_3	$cost$
a	a	(5,2)	a	a	(0,1)	a	a	(1,0)
a	b	(7,1)	a	b	(2,1)	a	b	**(5,5)**
b	a	(10,3)	b	a	(0,2)	b	a	(0,1)
b	b	(12,0)	b	b	(2,0)	b	b	**(1,1)**

TABLE 4.8: Cost table of MO-COP$_2$.

x_1	x_2	$cost$	x_2	x_3	$cost$	x_1	x_3	$cost$
a	a	(5,2)	a	a	**(3,3)**	a	a	**(4,4)**
a	b	(7,1)	a	b	(2,1)	a	b	(5,5)
b	a	**(2,2)**	b	a	(0,2)	b	a	(0,1)
b	b	**(3,0)**	b	b	(2,0)	b	b	(1,1)

4.2.0.1 Resilience in Dynamic MO-COP

In this subsection, two solution criteria for solving Dynamic MO-COP are introduced: *resistance* and *functionality*. Our focus is laid on a *reactive* approach, i.e., each problem MO-COP$_i$ in a DMO-COP can only be known at time step i ($0 \leq i \leq k$), and we have no information about the problems for any time step j where $j > i$. For dynamic problems, there exist two approaches, namely proactive and reactive. In a proactive approach, all problems in a DMO-COP are known in advance. Since we know all changes among problems, one possible goal of this approach is to find one (optimal) solution for a DMO-COP. On the other hand, in a reactive approach, since the new problem typically arises after solving the previous problem, it requires to solve each problem in a DMO-COP one by one. Thus, we need to find a sequence of Pareto front. In the following, the change of the constraint costs among problems in a DMO-COP is studied. [1]

Example 4.3. Consider a DMO-COP = $\langle$MO-COP$_0$, MO-COP$_1$, MO-COP$_2\rangle$. We use the same example represented in Example 2.3 and use it as the initial problem of this DMO-COP. The Pareto optimal solutions of MO-COP$_0$ are $\{(x_1, a), (x_2, a), (x_3, a)\}$ and $\{(x_1, a), (x_2, b), (x_3, b)\}$, and the Pareto front is $\{(6, 3), (10, 1)\}$ (see. Example 2.3). Table 4.7 shows the cost table of MO-COP$_1$. In Table 4.7, two constraints written in boldface are dynamically changed from the initial problem MO-COP$_0$, i.e., the cost vector of $\{(x_1, a), (x_3, b)\}$ and $\{(x_1, b), (x_3, b)\}$ are changed from $(1, 0)$ to $(5, 5)$ and from $(3, 2)$ to $(1, 1)$. The Pareto optimal solutions of MO-COP$_1$ are $\{(x_1, a), (x_2, a), (x_3, a)\}$ and $\{(x_1, b), (x_2, b), (x_3, b)\}$, and the Pareto front is $\{(6, 3), (15, 1)\}$. Table 4.8 represents the cost vector of MO-COP$_2$. In Table 4.8, four constraints written in boldface are additionally changed from MO-COP$_1$ i.e., $(2, 2)$, $(3, 0)$, $(3, 3)$, and $(4, 4)$. The Pareto optimal solutions of MO-COP$_2$ are $\{(x_1, b), (x_2, b), (x_3, a)\}$ and $\{(x_1, b),$

[1] Other changes, e.g., the number of variables, objectives and domain size, can be also considered. In this paper, the focus is laid on the dynamical change of constraint costs among problems. Similar assumption is also introduced in previous works [86, 87].

$(x_2, b), (x_3, b)\}$, and the Pareto front is $\{(3,3), (6,1)\}$. Thus, the solution of this DMO-COP is $\mathbb{PF} = \{\{(6,3), (10,1)\}, \{(6,3), (15,1)\}, \{(3,3), (6,1)\}\}$.

Now, two solution criteria for DMO-COPs are provided, namely, resistance and functionality. A sequence of assignments $\mathbb{A} = \langle A_0, A_1, ..., A_j\rangle$ is called an assignment trajectory, where A_i is an assignment of MO-COP$_i$ $(0 \leq i \leq j)$ and j is the current time step. Let m be the number of objectives and $R^h(A_i)$ be the cost for objective h obtained by assignment A_i $(1 \leq h \leq m)$, and l, q be constant vectors.

Definition 4.6 (Resistance)**.** For an assignment trajectory $\mathbb{A}$ and a constant vector $l = (l^1, l^2, ..., l^m)$, $\mathbb{A}$ is said to be l-resistant, iff for all h $(1 \leq h \leq m)$,

$$R^h(A_i) \leq l^h,\ (0 \leq i \leq |\mathbb{A}| - 1).$$

Definition 4.7 (Functionality)**.** For an assignment trajectory $\mathbb{A}$ and a constant vector $q = (q^1, q^2, ..., q^m)$, $\mathbb{A}$ is said to be q-functional, iff for all h $(1 \leq h \leq m)$,

$$for\ each\ j \in \{0, \ldots, |\mathbb{A}| - 1\}, \frac{\sum_{i=0}^{j} R^h(A_i)}{j+1} \leq q^h.$$

Resistance is the ability to maintain some underlining costs under a certain threshold, such that the system satisfies certain hard constraint and does not suffer from irreversible damages, i.e., the ability for a system to stay essentially unchanged despite the presence of disturbances. Functionality is the ability to maintain a guaranteed global quality for the assignment trajectory. While resistance requires to maintain a certain quality level at each problem in a DMO-COP, functionality requires to maintain this level in average, when looking over a certain horizon of time. Thus, functionality evaluates an assignment trajectory globally. The followings are two examples for them. We use the same example as in Example 4.3.

Example 4.4 (Resistance)**.** *The Pareto optimal solutions of the DMO-COP is* $\{(x_1, a), (x_2, a), (x_3, a)\}$ *and* $\{(x_1, a), (x_2, b), (x_3, b)\}$ *for MO-COP*$_0$*,* $\{(x_1, a), (x_2, a), (x_3, a)\}$ *and* $\{(x_1, b), (x_2, b), (x_3, b)\}$ *for MO-COP*$_1$*, and* $\{(x_1, b), (x_2, b), (x_3, a)\}$ *and* $\{(x_1, b), (x_2, b), (x_3, b)\}$ *for MO-COP*$_2$*. The corresponding Pareto front is* $PF_0 = \{(6,3), (10,1)\}$*,* $PF_1 = \{(6,3), (15,1)\}$*, and* $PF_2 = \{(3,3), (6,1)\}$*, respectively. Let* $l = (8,4)$ *be a constant vector. The assignment trajectory* $\mathbb{A} = \langle A_0, A_1, A_2\rangle$ *with* $A_0 = \{(x_1, a), (x_2, a), (x_3, a)\}$*,* $A_1 = \{(x_1, a), (x_2, a), (x_3, a)\}$*, and* $A_2 = \{(x_1, b), (x_2, b), (x_3, a)\}$ *is* l*-resistant, since* $R^1(A_0) = 6 < l^1\ (= 8)$*,* $R^2(A_0) = 3 < l^2\ (= 4)$*,* $R^1(A_1) = 6 < l^1$*,* $R^2(A_1) = 3 < l^2$*, and* $R^1(A_2) = 3 < l^1, R^2(A_2) = 3 < l^2$*. Similarly,* $\mathbb{A}' = \langle A_0, A_1, A_2'\rangle$ *is also* l*-resistant, where* A_0 *and* A_1 *are same as in* $\mathbb{A}$ *and* $A_2' = \{(x_1, b), (x_2, b), (x_3, b)\}$.

Algorithm 6 ASR

1: INPUT : DMO-COP = $\langle \text{MO-COP}_0, \text{MO-COP}_1, ..., \text{MO-COP}_\text{k} \rangle$ and two constant vectors $l = (l^1, l^2, ..., l^m)$, $q = (q^1, q^2, ..., q^m)$
2: OUTPUT : $\mathbb{RS}$ // set of sequences (all l-resistant and q-functional solutions)
3: $\mathbb{RS} \leftarrow \emptyset$
4: $l, q \leftarrow$ constant vectors
5: **for** each MO-COP$_i$ $(i = 0, ..., k)$ **do**
6: $\quad \mathbb{RS}_i^l \leftarrow ASR_{res}(\text{MO-COP}_\text{i}, l)$ // find all l-resistant solutions
7: $\quad$ **if** $\mathbb{RS}_i^l = \emptyset$ **then return** $\mathbb{RS} \leftarrow \emptyset$
8: $\quad$ **end if**
9: $\quad \mathbb{RS} \leftarrow \mathbb{RS} \otimes \mathbb{RS}_i^l$ // combine the current solution with previous solutions
10: $\quad \mathbb{RS}^q \leftarrow ASR_{fun}(\mathbb{RS}, q)$ // filter $\mathbb{RS}$ with q-functionality
11: $\quad$ **if** $\mathbb{RS}^q = \emptyset$ **then return** $\mathbb{RS} \leftarrow \emptyset$
12: $\quad$ **end if**
13: $\quad \mathbb{RS} \leftarrow \mathbb{RS}^q$
14: **end forreturn** $\mathbb{RS}$

Algorithm 7 ASR_{res}

1: INPUT : MO-COP$_i$ and l
2: OUTPUT : $\mathbb{RS}_i^l$ // a set of l-resistant solutions of MO-COP$_i$
3: $Root$: // the root of MO-COP$_i$
4: $AS \leftarrow \emptyset$ // an assignment of variables
5: $Cost \leftarrow$ null vector // cost vector obtained by AS
6: $\mathbb{RS}_i^l \leftarrow \emptyset$ // a set of pairs <cost vector, set of assignments>
// Launch solving from the root
7: $\mathbb{RS}_i^l \leftarrow first.solve(AS, Cost, \mathbb{RS}_i^l, l)$ **return** $\mathbb{RS}_i^l$

Example 4.5 (Functionality)**.** *Let* $q = (5, 4)$ *be a constant vector. The assignment trajectory* $\mathbb{A} = \langle A_0, A_1, A_2 \rangle$ *with* $A_0 = \{(x_1, a), (x_2, a), (x_3, a)\}$, $A_1 = \{(x_1, a), (x_2, a), (x_3, a)\}$, *and* $A_2 = \{(x_1, b), (x_2, b), (x_3, a)\}$ *is q-functional, since* $(6 + 6 + 3)/3 = 5 \leq q^1\ (= 5)$ *and* $(3 + 3 + 3)/3 = 3 < q^2\ (= 4)$. *However, for* $\mathbb{A}' = \langle A_0, A_1, A_2' \rangle$ *where* A_0 *and* A_1 *are same as in* $\mathbb{A}$ *and* $A_2' = \{(x_1, b), (x_2, b), (x_3, b)\}$, $\mathbb{A}'$ *is not q-functional, since* $(6 + 6 + 6)/3 = 6 > q^1\ (= 5)$.

4.2.0.2 Algorithm

An algorithm, *Algorithm for Systems Resilience* (ASR), for solving a DMO-COP is presented. This algorithm is based on the branch and bound search, which is widely used for COP and MO-COP algorithms, and it finds all resistant and functional solutions for DMO-COPs. Algorithm 6 shows the pseudo-code of ASR. The input is a DMO-COP that is a sequence of MO-COPs and constant vectors l and q. ASR outputs a set of sequences (all l-resistant and q-functional solutions). For each $MO\text{-}COP_i$ $(0 \leq i \leq k)$ in the sequence, ASR computes a set of all l-resistant solutions denoted $\mathbb{RS}_i^l$ by ASR_{res} (line 6). ASR uses the $\otimes$-operator and combines the set of sequences $\mathbb{RS}$ and the cost vectors of $\mathbb{RS}_i^l$ obtained by ASR_{res} (line 10). For example,

Algorithm 8 solve($AS, Cost, \mathbb{RS}_i^l, l$)

```
1: INPUT : < AS, Cost, RS_i^l, l >
2: OUTPUT : RS_i^l
3: for each value v_1 of the variable domain do
4:     AS ← v_1
5:     local_cost ← null vector
   // step 3.1: Compute local cost of the choice
6:     for each constraint with an ancestor a do
7:         v_2 ← value of a in AS
8:         local_cost ← local_cost + cost(v_1, v_2)
9:     end for
10:    new_cost ← Cost + local_cost
   // step 3.2: Bound checking
11:    dominated ← F
12:    for each objective h (1 ≤ h ≤ m) do
13:        if r^h > l^h, r^h ∈ new_cost then
14:            AS ← (AS \ v_1)
15:            dominated ← T
16:        end if
17:    end for
18:    if new_cost is dominated by RS_i^l then
19:        AS ← (AS \ v_1)
20:        dominated ← T
21:    end if
22:    if dominated then
23:        continue
24:    end if
   // step 3.3.1: New Pareto optimal solution
25:    if AS is complete then
26:        E ← all elements of RS_i^l dominated by new_cost
27:        RS_i^l ← RS_i^l \ E
28:        RS_i^l ← RS_i^l ∪ {(new_cost, AS)}
29:        continue
30:    end if
   // step 3.3.2: Continue solving
31:    RS_i^l ← next.solve(AS, Cost, RS_i^l, l)
32:    AS ← (AS \ v_1)
33: end forreturn RS_i^l
```

after the combination of $\mathbb{RS} = \{(6,3),(10,1)\}$ and $\mathbb{RS}_1^l = \{(6,3),(15,1)\}$, i.e., $\mathbb{RS} \otimes \mathbb{RS}_1^l$, there exist four sequences $\{\{(6,3)\},\{(6,3)\}\}$, $\{\{(6,3)\},\{(15,1)\}\}$, $\{\{(10,1)\},\{(6,3)\}\}$ and $\{\{(10,1)\},\{(15,1)\}\}$. For the initial problem, i.e., $MO\text{-}COP_0$, $\mathbb{RS}$ is equal to $\mathbb{RS}_0^l$ when $\mathbb{RS}_0^l \neq \emptyset$. Next, ASR checks the q-functionality of each sequence of $\mathbb{RS}$ (line 11). Finally, it provides a set of all l-resistant and q-functional solutions if there exists. Otherwise, it outputs the empty set (checked in line 7-9 and line 12-14). In case l and q are large enough (i.e., no restriction), all Pareto optimal solutions become l-resistant and q-functional solutions in the worst case, and the size of $\mathbb{RS}$ finally becomes $|\mathbb{RS}_0| \times |\mathbb{RS}_1| \times ... \times |\mathbb{RS}_k|$.

Algorithm 9 ASR_{fun}

1: INPUT : $\mathbb{RS}, q$
2: OUTPUT : $\mathbb{RS}^q$ // set of the filtered sequences
3: $\mathbb{RS}^q \leftarrow \emptyset$
4: **for** each sequence $S_j \in \mathbb{RS}$ $(0 \leq j \leq |\mathbb{RS}| - 1)$ **do**
5: **for** each h $(1 \leq h \leq m)$ **do**
6: **if** $\frac{\sum_{i=0}^{|S_j|-1} s_{i,j}^h}{|S_j|} > q^h$ // $s_{i,j} \in S_j$ **then return** $\mathbb{RS}^q = \emptyset$
7: **end if**
8: **end for**
9: $\mathbb{RS}^q \leftarrow S_j$
10: **end forreturn** $\mathbb{RS}^q$

Let us describe ASR_{res}. It finds all Pareto optimal solutions of each problem in a sequence, which are bounded by the parameter l. Algorithm 7 and 8 represent the pseudo-codes of ASR_{res}. We assume that a variable ordering, i.e., pseudo-tree [26], is given. The input is an $MO\text{-}COP_i$ $(0 \leq i \leq k)$ and a constant vector l, and the output is the entire set of l-resistant solutions (line 1 and 2 in Algorithm 7). ASR_{res} starts with an empty set of l-resistant solutions and a null cost vector, and solves the first variable according to the variable ordering (line 3-7). It chooses a value for the variable and updates the cost vector according to the cost tables (step 3.1 in Algorithm 8). At this moment the obtained cost vector has to ensure the following two properties: (i) r^h (the cost for objective h) is bounded by the constant vector l and (ii) the cost vector is not dominated by another cost vector (i.e., current l-resistant solutions) in $\mathbb{RS}_i^l$. If one of the two properties is violated, ASR_{res} branches on the next value of the variable. When its domain is completely explored, the search branches to the previous variable and continues the solving (step 3.2 in Algorithm 8). When all assignments are formed, i.e., no variable left to be assigned, a new solution is added to $\mathbb{RS}_i^l$. All previous dominated solutions are removed from $\mathbb{RS}_i^l$ and the search continues with the next values of the variable (step 3.3.1 in Algorithm 8). In case r^h fulfills the two properties, it continues the solving with the next variable according to the variable ordering (step 3.3.2). The search stops when the whole search space has been covered by branch and bound search. ASR_{res} finally outputs the set of l-resistant solutions (which are not dominated by another solutions) [2].

Let us describe ASR_{fun}. Algorithm 9 shows the pseudo-code of it. The input is a set of sequences obtained by ASR_{res} and a constant parameter q, and output is a set of l-resistant and q-functional solutions $\mathbb{RS}^q$ (line 4-11). For each sequence S_j of $\mathbb{RS}$, it checks the q-functionality by using the equation given in definition 4.7 (line 5-9). ASR is a complete algorithm, i.e., it provides a set of all l-resistant and q-functional solutions if there exists. Otherwise, it outputs empty set (line 7-9 and 12-14 in Algorithm 6).

[2] ASR_{res} checks the dominance among the solutions (in line 18, 22 and 26 in Algorithm 8) and provides the set of Pareto optimal solutions bounded by the parameter l.

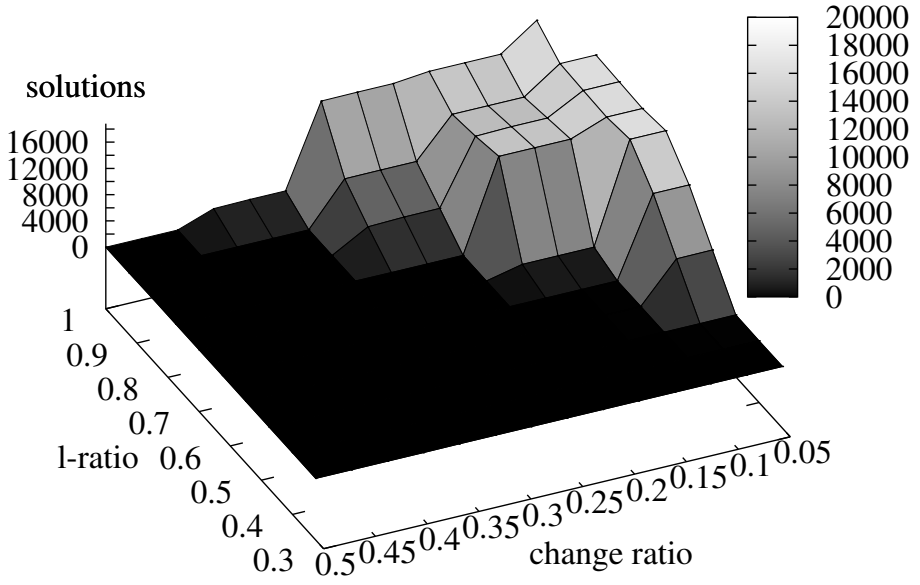


FIGURE 4.1: Number of solutions.

4.2.1 Experimental Results

In this subsection, the performances of ASR are evaluated with different types of dynamical changes. In the experiments, we generate DMO-COPs that contain three MO-COPs as in Example 4.3. All the tests are made with 20 variables, the domain size of each variable is two, the number of objectives is two, and the cost of each constraint is chosen uniformly at random from the range [0:100]. In DMO-COPs, we change a fixed proportion of constraint costs (called the change ratio) at each dynamic step. For the initial problem, we choose the constraint costs from [0:100]. Then, we create the next problem by changing the proportion of constraints costs defined by the change ratio. For example, in case the change ratio is 5%, we choose 5% of all constraints in the current problem and change their constraint costs by selecting the new costs from the range [100:200], but do not change the remaining constraint costs. Each data point in a graph represents the average of 50 problem instances.

Figure 4.1 and 4.2 show the average number of solutions and its runtime obtained by ASR. The l-ratio is provided by $l/(cost_{max} \times \#constraints)$, where $cost_{max}$ is the maximal cost value (i.e., 200). We vary the the change ratio from 0.05 to 0.5 and from 0.3 to 1.0 for l-ratio. In this experiment, we set the constant vector to $q = (q_{max}, q_{max})$, where $q_{max} = 3 \times (cost_{max} \times \#constraints)$ which is large enough, so that all l-resistant solutions become q-functional. In figure 4.1, we can observe that only small change ratio (i.e., from 0.05 to 0.15) for the l-ratio from 0.5 to 1.0 allows us to find resistant (and also functional) solutions for DMO-COPs. On the other hand, in case the l-ratio is small and the change ratio is large, ASR

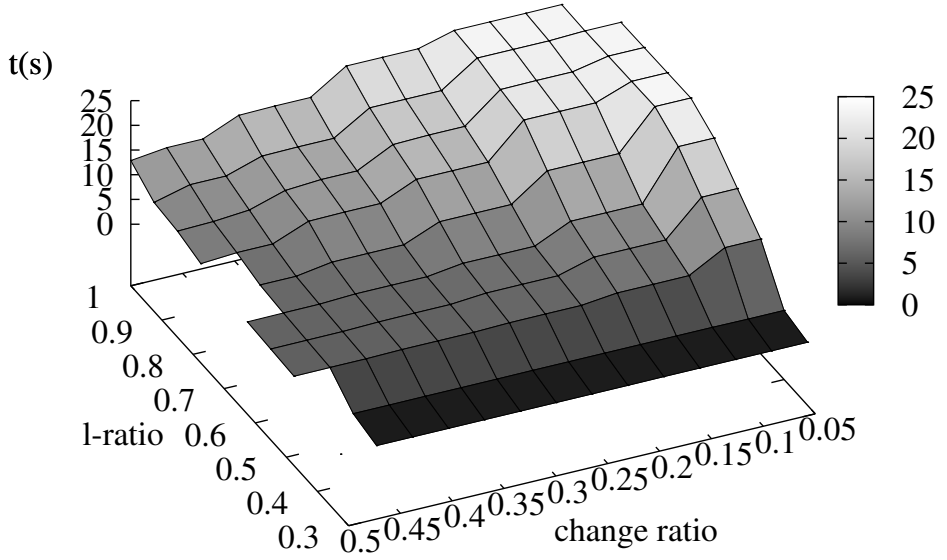
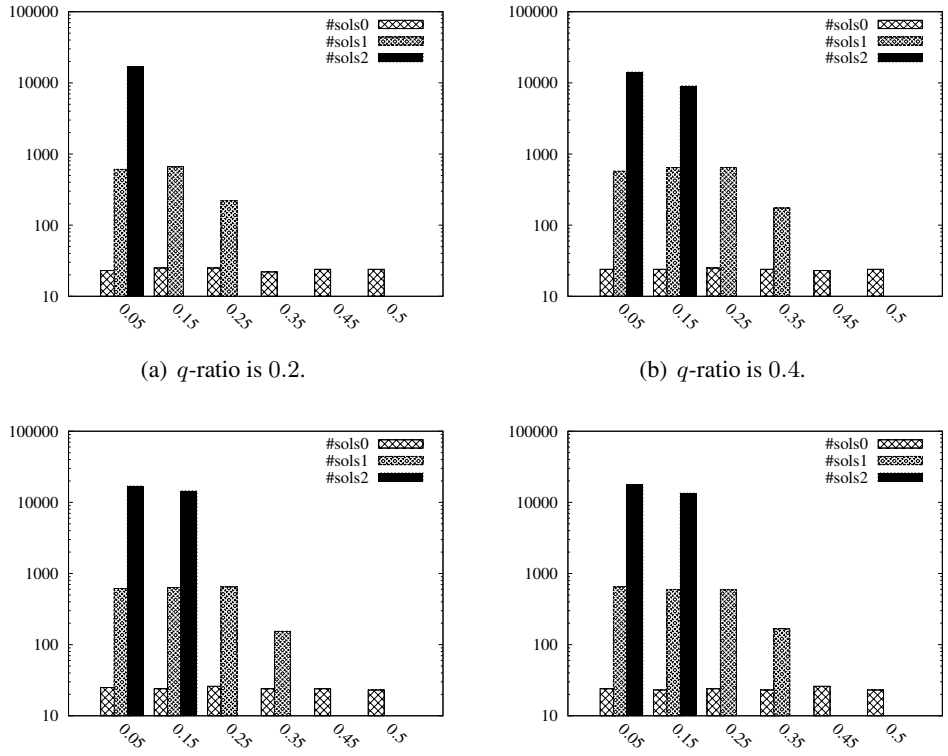


FIGURE 4.2: Runtime.

cannot find any solutions. In figure 4.2, we observed that the average runtime increases where the number of resistant (and also functional) solutions becomes large.

Figure 4.3 shows the results for the average number of obtained resistant and functional solutions of each problem in DMO-COP, i.e., MO-COP$_0$, MO-COP$_1$ and MO-COP$_2$, for varying the q-ratio. In this experiment, we set l-ratio to 0.8. The $\#sols0$ represents the number of obtained resistant and functional solutions in MO-COP$_0$, $\#sols1$ is for MO-COP$_1$, and $\#sols2$ shows that for MO-COP$_2$. The x axis shows the change ratio and the y axis is the number of solutions obtained by ASR. For any q-ratio, we find solutions for the initial problem (i.e., MO-COP$_0$) in DMO-COPs. It is the problem where Pareto optimal solutions have the lowest cost vectors. Once this problem changes (with regard to the change ratio), the average cost vector increases and the functionality becomes harder to obtain. For small changes where the change ratio is 0.05, even a low q-ratio ($q = 0.2$) allows to find resistant and functional solutions (Figure 4.3(a)). However, for more drastic changes, e.g., the change ration is 0.15, we need a higher q-ratio in order to find solutions after the third problem. We then reach a point where the change ratios are too severe (i.e., 0.25-0.5) to find solutions for the third problem. We can increase the q-ratio but we cannot find resistant and functional solutions (Figure 4.3(b)- 4.3(d)).

In summary, these experimental results reveal that the performance of ASR is highly influenced by change ratio. ASR can obtain the resistant and functional solutions of a DMO-COP, when the dynamical changes are small (i.e. the change ratio is from 0.05 to 0.15). Otherwise, ASR outputs empty set quickly.



(a) q-ratio is 0.2.

(b) q-ratio is 0.4.

(c) q-ratio is 0.6.

(d) q-ratio is 0.8.

FIGURE 4.3: The average number of resistant and functional solutions of each problem in DMO-COPs.

4.2.2 Conclusion

In this section, two solution criteria for DMO-COPs have been defined, namely, resistance and functionality, which are properties of interest underlying the *resilience* for DMO-COPs. An algorithm called ASR for solving a DMO-COP has been presented and evaluated. ASR aims at computing every resistant and functional solution for DMO-COPs.

As a perspective for further research, we intend to apply our approach to some real-world problems, especially dynamic sensor network and scheduling problems, and will develop algorithms that are specialized to these application problems (by modifying ASR). Furthermore, we will investigate some methods for choosing solutions like [88] introduced, and will apply them in our framework.

4.3 Limiting Transition in Dynamic Constraint Optimization Problems

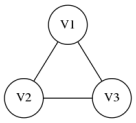
In many applications of Distributed Constraint Optimization Problems, the problem changes over time. Several works have considered Dynamic DCOP [87, 89–91] that can usually be represented using a sequence of static DCOPs. Some of these works focus on reacting to changes by finding a new solution as efficiently as possible, reusing previously gathered information to speed-up the search for an optimal assignment. Another approach is to find the best possible solution in a given amount of time, which is important for applications where a decision should be made before some deadline. One aspect that is mostly overlooked however is the cost of adopting a new solution. The few works on Dynamic DCOP that mention such a cost assume that it can simply be included in the utility function of the problem. A more recent work proposed a definition of a Proactive Dynamic DCOP [92] that includes random variables to help make decision in advance of the changes over a finite horizon. This model includes a transition cost corresponding to the cost of changing the assignment of a variable between time steps.

The transition cost represents the price to pay to switch from a previous solution to a new one. In many applications, this might drain some limited resources (money, energy, ...) that should be spent carefully. This can be seen as a new objective to optimize, creating trade-offs between solution quality and transition cost.

In this section, we present a model called Transition-Sensitive Dynamic DCOP where the transition cost is considered as a new objective to optimize. We then present the Limited Transition Problem which considers a limit on how much transition cost is allowed for the new solution while still optimizing its utility. This can be used when we have a real resource limit, when we want to avoid spending all our resources at once in case more changes happen in the future, or even for convenience when we do not want to drastically change a solution. Such an approach can be applied to logistic, when rerouting trucks to service additional clients [33] with limited fuel, or to scheduling problems [38] where we do not want to make too many changes to a schedule.

This section is structured as follows. In 4.3.1, we propose the definition of a Transition-Sensitive Dynamic DCOP (TS DynDCOP) and introduce the Limited Transition Problem (LTP) as a way to tackle a TS DynDCOP in a reactive fashion. In 4.3.2, we propose an incomplete and a complete approach to solve the LTP.In 4.3.3, we evaluate our algorithms on instances of the dynamic meeting scheduling problem. In 4.3.4, we summarize our contributions and present some ideas for future works.

FIGURE 4.4: Example of Distributed Constraint Optimization Problem.



V_1	V_2	$reward$	V_1	V_3	$reward$	V_2	V_3	$reward$
0	0	2	0	0	1	0	0	1
0	1	1	0	1	1	0	1	2
1	0	0	1	0	0	1	0	1
1	1	2	1	1	3	1	1	1

4.3.1 Transition Costs in Dynamic DCOP

In this subsection, we present the Transition Sensitive Dynamic DCOP which adds transition costs to the Dynamic DCOP model, effectively turning it into an MO-DCOP. To use this new model in a reactive way, we propose the Limited Transition Problem (LTP) which takes as argument a limit on how much transition cost is allowed. We then show a quality guarantee for solutions of the LTP when the transition cost considered is the number of changed variables.

We consider that the cost of adopting a new solution A' depends on the previous solution A, and we denote the corresponding transition cost function by $\delta(A, A')$. Various transition cost functions can be considered depending on the problem, the most basic being the Hamming distance $\mathcal{H}$, commonly used in scheduling and timetabling problems, which measures the number of changed variables.

4.3.1.1 Transition-Sensitive Dynamic DCOP

We now propose the definition of a Transition-Sensitive Dynamic DCOP (TS DynDCOP) where adopting an assignment has a cost that needs to be taken into account.

Definition 4.8. A Transition-Sensitive Dynamic DCOP (TS DynDCOP) is a tuple (Seq, Δ) where Seq is a sequence $\langle DCOP_0, DCOP_1, \ldots, DCOP_l \rangle$, and Δ is a set of transition cost function $\{\delta_0, \delta_1, \ldots, \delta_l\}$.

Each $DCOP_i = (X^i, V^i, D^i, C^i, F^i)$ is associated to a transition cost function $\delta_i : \times_{\forall v_j in V^i} D^i_j \rightarrow \mathbb{R}$ that measures the transition cost of adopting an assignment for the problem $DCOP_i$. The goal is then to find an assignment A_i for each problem $DCOP_i$ such that it (i) optimizes the solution's utility $R(A_i)$ and (ii) minimizes the transition cost $\delta_i(A_i)$.

A TS DynDCOP can be seen as a more general definition of the common Dynamic DCOP (Definition 2.11) where an additional objective function is considered to model the cost of adopting a new assignment.

4.3.1.2 Limited Transition Problem

We now propose the *Limited Transition Problem* (LTP) that can solve a TS DynDCOP in a reactive way. In the LTP, we are given an initial assignment α and a maximum allowed transition cost d, and want to find the best solution within this limit.

Definition 4.9. The Limited Transition Problem (LTP) is defined as a tuple $\Pi = (\Theta, \alpha, \delta, d)$, where Θ is a Distributed Constraint Optimization Problem, α is an assignment for Θ that is called initial assignment, δ is a function that defines a distance between two assignments, and d is the maximum acceptable distance between two assignments.

A solution to an LTP is an assignment A for Θ such that (i) $\delta(\alpha, A) \leq d$ and (ii) there does not exist another assignment A' such that $\delta(\alpha, A') \leq d$ and $R(A') \succ R(A)$.

Example 4.6. Let us consider the DCOP of Figure 4.4 where $A = \{(V_1, 1), (V_2, 1), (V_3, 1)\}$ is the solution. Now let us imagine the dynamic case where the constraint between V_1 and V_3 was removed, generating Θ, the new DCOP to solve.

We consider the resulting Limited Transition Problem with initial assignment $\alpha = \{(V_1, 1), (V_2, 1), (V_3, 1)\}$ and $\delta = \mathcal{H}$, the Hamming distance.

We will now evaluate the solution of this LTP for different values of d. The case where $d = 0$ is straightforward. Since we allow no change from the initial assignment, the solution of the LTP is α whose reward is now $R(\alpha) = 3$.

For $d = 1$, we need only to consider, in addition to α, assignments where one variable has a different value compared to α: $\{(V_1, 1), (V_2, 1), (V_3, 0)\}$ yields a reward of 3, $\{(V_1, 1), (V_2, 0), (V_3, 1)\}$ yields a reward of 2 and $\{(V_1, 0), (V_2, 1), (V_3, 1)\}$ yields a reward of 2. Since $R(\alpha) = 3$, the LTP can have two solutions for $d = 1$, $\{(V_1, 1), (V_2, 1), (V_3, 0)\}$ or α.

Finally, let us consider $d = 2$, allowing two variables to change. The new assignments to consider (in addition to α and the one we considered for $d = 1$) are: $\{(V_1, 1), (V_2, 0), (V_3, 0)\}$ with a reward of 1, $\{(V_1, 0), (V_2, 1), (V_3, 0)\}$ with a reward of 2, and $\{(V_1, 0), (V_2, 0), (V_3, 1)\}$ with a reward of 4. The solution of the LTP for $d = 2$ is thus $\{(V_1, 0), (V_2, 0), (V_3, 1)\}$.

4.3.1.3 Quality Guarantee

When $\delta = \mathcal{H}$, we can provide a guarantee on the quality of the solution obtained for the Limited Transition Problem. This guarantee is presented as a ratio of the optimal solution that we would obtain if we did not limit the transition cost (equivalent to $d = \infty$) and is a generalization of the guarantee defined for k-size optimality [93].

k-size optimality is a solution criteria for DCOP where an assignment is k-size optimal if its reward cannot be improved by changing the values of k or less of its variables. The value of k can vary between 0 (all solutions are 0-size optimal) and $|V|$, the number of variables in the problem (only an optimal solution of the DCOP is $|V|$-size optimal). When considering the transition cost as the Hamming distance between the two assignments ($\delta = \mathcal{H}$), there exists a strong relationship between the solution of a Limited Transition Problem and a k-size optimal solution.

Property 9. A k-size optimal solution A of the problem Θ is a solution of the LTP $\Pi = (\Theta, A, \mathcal{H}, k)$.

Proof. A is already within the required transition cost k since $\mathcal{H}(A, A) = 0$ and $k \geq 0$. For A to not be solution of the LTP, there must be a solution A' such that $\mathcal{H}(A, A') \leq k$ and $R(A') \succ R(A)$. However, since A is k-size optimal, any solution within a distance k cannot have a better utility, making A solution of the LTP. □

We now provide a generalization of the guarantee of k-size optimality that we can use a priori for the LTP.

Property 10. For a Limited Transition Problem $\Pi = (\Theta, \alpha, \mathcal{H}, d)$ where Θ is a *maximization* DCOP with n variables and a maximum constraint arity of m, we can express the following relation between the reward of the LTP solution A and the reward of the optimal solution A^o of Θ:

$$R(A) \geq \frac{\binom{n-m}{d-m}}{\binom{n}{d} - \binom{n-m}{d}} R(A^o) \tag{4.1}$$

Proof. Consider a LTP $\Pi = (\Theta, \alpha, \mathcal{H}, d)$ whose solution is the assignment A. By definition, any assignment $\tilde{A}$ such that $\mathcal{H}(\alpha, \tilde{A}) \leq d$ must have a reward $R(\tilde{A}) \leq R(A)$. We call this set of assignment $\tilde{\mathcal{A}}$. Now consider any non-null subset $\hat{\mathcal{A}} \subset \tilde{\mathcal{A}}$. For any assignment $\hat{A} \in \hat{\mathcal{A}}$, the constraints C in the DCOP θ can be divided into three discrete sets, given α and $\hat{A}$:

- $C_1(\alpha, \hat{A}) \subset C$ that includes only the constraints between variables in $\hat{A}$ which have deviated from their values in α.
- $C_2(\alpha, \hat{A}) \subset C$ that contains only the constraints between variables that did not deviate from α.
- $C_3(\alpha, \hat{A}) \subset C$ that contains the remaining constraints between at least one deviating variable and one non-deviating variable.

The reward of an assignment $\hat{A}$ can be written as:

$$R(\hat{A}) = \sum_{c \in C_1(\alpha, \hat{A})} R_c(\hat{A}) + \sum_{c \in C_2(\alpha, \hat{A})} R_c(\hat{A}) + \sum_{c \in C_3(\alpha, \hat{A})} R_c(\hat{A}) \tag{4.2}$$

It was shown in [93] that we can express the sum of rewards from all assignments $\hat{A} \in \hat{\mathcal{A}}$ as follows:

$$\sum_{\hat{A} \in \hat{\mathcal{A}}} R(\hat{A}) \geq \sum_{\hat{A} \in \hat{\mathcal{A}}} \sum_{c \in C_1(\alpha, \hat{A})} R_c(\hat{A}) + \sum_{\hat{A} \in \hat{\mathcal{A}}} \sum_{c \in C_2(\alpha, \hat{A})} R_c(\hat{A}) \tag{4.3}$$

And since $R(A) \geq R(\hat{A}), \forall \hat{A} \in \hat{\mathcal{A}}$, we have:

$$R(A) \geq \frac{\sum_{\hat{A} \in \hat{\mathcal{A}}} (\sum_{c \in C_1(\alpha, \hat{A})} R_c(\hat{A}) + \sum_{c \in C_2(\alpha, \hat{A})} R_c(\hat{A}))}{|\hat{\mathcal{A}}|} \tag{4.4}$$

We now consider the subset $\hat{\mathcal{A}}$ such that every $\hat{A} \in \hat{\mathcal{A}}$ have d variables that changed from α and these variables take their values from A^o. For the denominator, $|\hat{\mathcal{A}}| = \binom{\mathcal{H}(\alpha, A^o)}{d}$. Then, for every constraint $c \in C$, there are exactly $\binom{\mathcal{H}(\alpha, A^o) - |c|}{d - |c|}$ different assignments $\hat{A} \in \hat{\mathcal{A}}$ for which $c \in C_1(\alpha, \hat{A})$. Similarly, there are exactly $\binom{\mathcal{H}(\alpha, A^o) - |c|}{d}$ different assignments for which $c \in C_2(\alpha, \hat{A})$. Finally, we consider the worst case where $\mathcal{H}(\alpha, A^o) = n$ and $|c| = m$ and obtain Equation 4.1. □

Figure 4.5 shows the ratio of the optimal quality that is guaranteed for different numbers of variables, binary constraints and varying the parameter d. There is no guarantee when $d \leq 2$ but for $d > 2$, we can see an almost linear growth of the guarantee, reaching optimality guarantee when d is equal to the number of variables. While this guarantee can be interesting in some application where a critical level of quality has to be met, the transition cost allowed d has to be set to a high value before a good quality guarantee can be reached.

4.3.2 Algorithms for the LTP

In this subsection, we propose some approaches to solve the Limited Transition Problem. Limiting the transition cost allowed is equivalent to putting a bound on one of the objective. We can thus expect lower values of d to reduce the time required to find a solution. This can easily be shown by expressing the number of assignments within a distance of d when using the Hamming distance $\mathcal{H}$ as our transition cost function.

Property 11. For a Limited Transition Problem $\Pi = (\Theta, \alpha, \mathcal{H}, d)$ where Θ is a DCOP with n variables of domain size $|D|$, we can express the number of assignments for Θ that are exactly at a distance d of α as:

$$\binom{n}{d} (|D| - 1)^d \tag{4.5}$$

Proof. Given an assignment A of n variables, there are $\binom{n}{d}$ subsets $\hat{A} \in A$ of size d. Now, considering any assignment $\hat{A}$ of d variables of domain D, there are exactly $(|D| - 1)^d$ possible assignments where each variable has a different value than in $\hat{A}$, giving us Equation 4.5. □

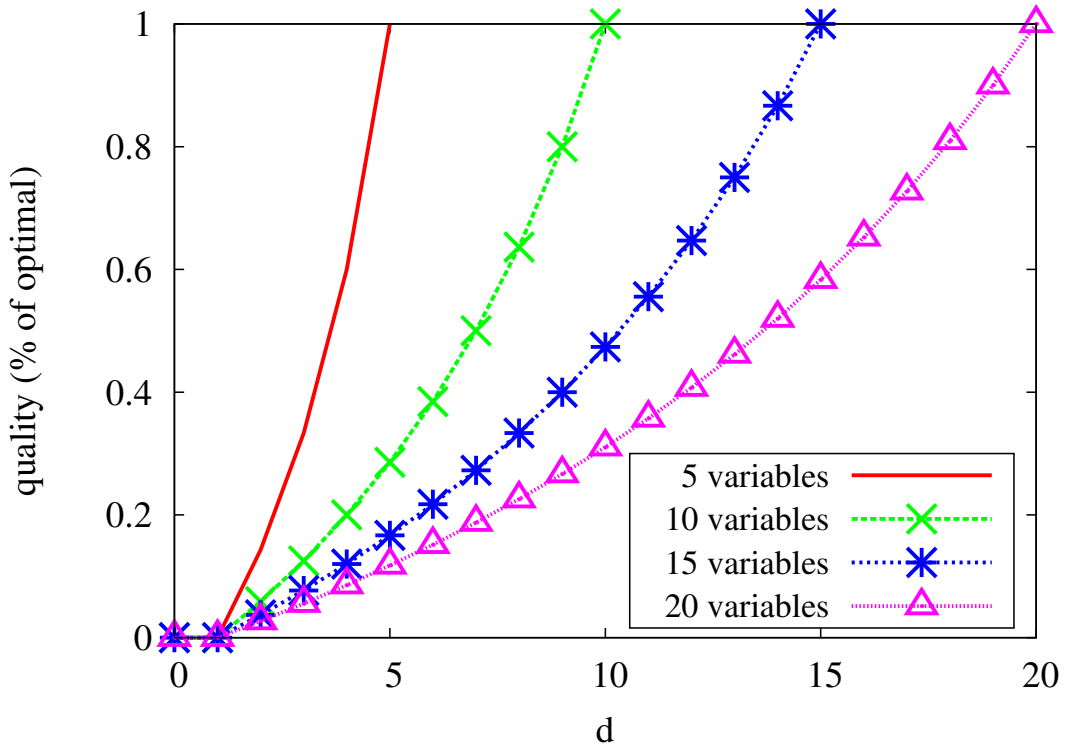


FIGURE 4.5: Quality bounds for solutions of the LTP varying d and with binary constraints.

Corollary 4.10. *For a Limited Transition Problem* $\Pi = (\Theta, \alpha, \mathcal{H}, d)$ *where* Θ *is a DCOP with* n *variables of domain size* $|D|$*, we can express the number of assignments for* Θ *that are within a distance* d *from* α *as:*

$$\sum_{i=0}^{d} \binom{n}{i} (|D| - 1)^i \tag{4.6}$$

The number of assignments within distance d of α is, in the worst case, the total number of assignment we have to check to find the solution of the LTP. This can give us an idea on the impact of d on the search space. , and we show in Figure 4.6 the number of assignments based on d for different number of variables with a domain size of 2.

For all the methods we propose in this subsection, we will make several assumptions. First, we assume *superadditive* transition cost functions such that changing additional variables never decreases the transition cost, i.e., if $\mathcal{H}(A, A') > \mathcal{H}(A, A'')$, then $\delta(A, A') \geq \delta(A, A'')$. We also assume that $\delta(A, A) = 0$, i.e., not changing an assignment induces no transition cost. Second, we consider that an agent x_i is responsible of one variable v_i. To communicate during the various algorithms, we use a widely popular graph structure for DCOP algorithms called pseudo-tree [26]. In a pseudo-tree, there exists a unique root node, and each non-root node has a parent node. Depending on the requirement of the algorithms, additional constraints on the pseudo-tree can exist. Most commonly, it might be required that all variables sharing a constraint

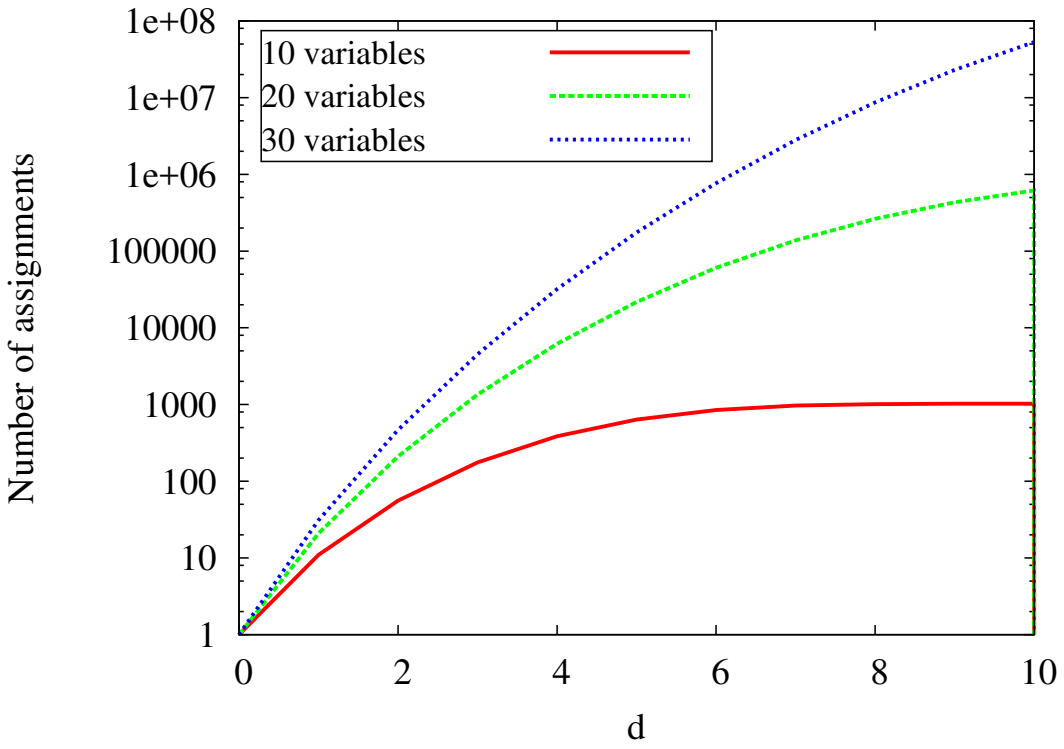


FIGURE 4.6: Number of possible assignments within distance d of α

must be part of each others' ancestors or descendants, and such structure can be obtained using a depth-first traversal of the constraint graph.

4.3.2.1 Complete Multi-Objective Search

The first method we consider is a complete multi-objective search that can find all trade-offs between the solution quality and the transition cost. Such method was proposed to solve a Multi-Objective DCOP [94] and, when applied to solving the LTP, guarantees to find its optimal solution.

The basic idea of such a complete multi-objective search is for each agent in the pseudo-tree to compute a local Pareto front for itself and its descendants, given an assignment of values of its ancestors. To find this local Pareto front, partial assignments are passed-down the tree, with each variable adding its own current value to the partial assignment and sending it in turn to its children. When a leaf receives a partial assignment, it computes the corresponding utility and sends it to its parent. Thanks for the depth-first structure we previously mentioned, a leaf node can compute the utility of *all* its constraints by knowing the current values of its ancestors. After receiving the utility of all its children, a node will update the partial assignment with another value from its domain and send it to its children. Once all values have been explored, a node can compute the complete set of local trade-offs for itself and its descendants by combining its

own local utility with the one receives from its children. Once every combination of values have been explored, the root node will know the complete Pareto front of the problem.

For the LTP, we can add a way to prune parts of the search space. When a partial assignment A is sent down the pseudo-tree, we check if we are already paying too much transition cost. If $\delta(\alpha, A) > d$, we can abandon the partial assignment since it can never produce a solution within the transition cost limit of d. This is true because we assume *superadditive* transition cost function. For more general functions, we would not be able to perform this pruning.

4.3.2.2 Limited Local Search

We now describe a local search for solving the LTP. Since we are trying to find a solution within a given distance from an initial assignment α, a local search that changes variables' values one at a time appears as a natural method for tackling the LTP. We thus propose the Limited Local Search (LLS) and show the corresponding pseudo-code in Algorithm 10. Because of the importance of respecting the parameter d, the neighborhood considered during the LLS depends on the transition cost of the current assignment. If the current assignment has a transition cost below d, then the LLS behaves like a traditional local search, exploring a neighborhood that provides the best solution quality. If the transition cost is above d however, only changes that reduce the transition cost are considered.

Each agent starts by searching for its best possible new value. val_i is initialized to its value in the current assignment (line 6). Then, it tries each value $val \in D_i$ and consider the assignment A' that would result in assigning val to variable v_i. Three tests are then made to decide if this value is the best tried so far. Firstly, in order to avoid falling into a local optima, the agent makes sure A' is not part of the previously explored assignments PA. Secondly, the agent tests if val provides a better utility than the current val_i (line 10). Finally, it tests if the current assignment A has a transition cost higher than d, and if so, it makes sure that the new value reduces this cost. Once these tests are performed for all possible values of the domain, the agent knows which new value is best for v_i. It now needs to determinate the best change between his own new value and the ones received from its children. To do so, we propose a measure of the improvements which varies between maximization (line 19) or minimization (line 21) of the utilities. Since each agent only knows the constraints associated with its own variable, the only way to compare the changes from different variables is by using this improvement measure. Once an agent has received a potential change from each of its children (line 24), it selects the one with the best improvement and sends it to its parent (line 32). Once this step is reached by the root, it knows the best change between all variables and can send it to all its descendants, allowing each agent to update its knowledge of the current assignment. In addition, since we allow to lower the quality of the current assignment (when lowering the transition cost), each agent should remember the

Algorithm 10 Algorithm of LLS for an agent x_i responsible of variable v_i

1: C^i: set of constraints including v_i.
2: A: the current values of all variables.
3: PA: the set of previous assignments explored.
4: A^+: the best assignment found to far.
5: **while** termination condition not met **do**
6: $val_i \leftarrow A_i$
7: **for** each $val \in D_i$ **do**
8: $A' \leftarrow A \cup (v_i, val)$
9: **if** $A' \notin PA$ **then**
10: **if** $\sum_{c \in C^i} R_c(A') \succ \sum_{c \in C^i} R_c(A \cup (v_i, val_i))$ **then**
11: **if** $\delta(\alpha, A) \leq d \vee \delta(\alpha, A) < \delta(\alpha, A')$ **then**
12: $val_i \leftarrow val$
13: **end if**
14: **end if**
15: **end if**
16: **end for**
17: $A' \leftarrow A \cup (v_i, val_i)$
18: **if** maximization **then**
19: $improvement_i \leftarrow \sum_{c \in C^i} R_c(A') - R_c(A)$
20: **else**
21: $improvement_i \leftarrow \sum_{c \in C^i} R_c(A) - R_c(A')$
22: **end if**
23: **for** each child v_j **do**
24: receive $(improvement_j, v_j, val_j)$ from v_j
25: **end for**
26: $best \leftarrow \text{argmax}_j \ improvement_j$
27: $PA \leftarrow PA \cup A$
28: **if** v_i is root **then**
29: $A \leftarrow A \cup (v_{best}, val_{best})$
30: send A to all children
31: **else**
32: send $(improvement_{best}, v_{best}, val_{best})$ to parent
33: receive A from parent
34: **end if**
35: **if** $R(A) \succ R(A^+) \ \wedge \ \delta(\alpha, A) \leq d$ **then**
36: $A^+ \leftarrow A$
37: **end if**
38: **end while**

best assignment A^+ encountered so far[3] (line 36). This is repeated until an end condition have been met, which usually involves a limit on the time or on the number of iterations.

4.3.2.3 Starting from A^o

The Local Search algorithm we previously proposed is originally made to start from α, whose transition cost is 0. However, it is also possible to start from an assignment whose transition cost

[3]Line 35, we used $R(A)$ for simplicity but in practice, we maintain a sum of all implemented improvements to determine A^+.

	Time (s)						
	d=0	d=1	d=2	d=3	d=4	d=5	d=6
Complete	0.48	1.8	6.7	27	89	220	294
LLS-α	5						
LLS-A^o							
	Cost						
	d=0	d=1	d=2	d=3	d=4	d=5	d=6
Complete	4.95	3.45	2.45	1.80	1.25	1.25	1.23
LLS-α	7.05	4.85	3.50	2.50	1.85	1.55	1.45
LLS-A^o	6.00	3.65	2.50	1.90	1.45	1.40	1.40

TABLE 4.9: Results for solving the LTP on instances of meeting scheduling (10 meetings)

is *over* the limit of d. In this case, the Limited Local Search works by first reducing d, selecting at each step a new value that can reduce d for the minimum loss of quality. Once we reach an assignment whose transition cost is below d, LLS no longer focuses solely on reducing d and, like we describe earlier, tries to improve the current assignment.

We propose to start LLS from the optimal solution A^o of the DCOP Θ. We can expect that in cases where d is high, a local search starting from A^o will yield a better quality than starting from α. The main problem with such approach is that it requires to solve Θ, which can be done using any existing DCOP algorithm and might take a lot of time. However, even when we have to first solve Θ, it might still be a faster approach compared to the complete multi-objective search.

4.3.3 Experimental Evaluation

In this subsection, we compare the different proposed methods on various dynamic scenarios. When using α (resp. A^o) as the starting assignment for our local search algorithm, we will refer to LLS-α (resp. LLS-A^o). For our experiments, we used Java 1.7 using the Jade agent framework [95] on an Intel Xeon X5650 machine with 6 core at 2.67GHz and 12Go of RAM.

We experimented with random instances of the distributed meeting scheduling problem [38] where a set of agents $\mathcal{A}$ want to attend several meetings $\mathcal{M}$ within a set of time periods $\mathcal{T}$. A mapping $\mathcal{S} : \mathcal{M} \rightarrow \mathcal{A}$ indicates the subset of agents wanting to attend each meeting. We consider the events as variables (EAV) formulation where each meeting $m_i \in \mathcal{M}$ is represented as a variable v_i with domain $D_i = \mathcal{T}$ and constraints between variables indicate that they should be assigned different times. For each pair of meetings $m_i, m_j \in \mathcal{M} \times \mathcal{M}$s.t.$m_i < m_j$, let us denote $\mathcal{A}^{i,j}$ the set of agents wanting to attend both m_i and m_j ($\mathcal{A}^{i,j} = \mathcal{S}(m_i) \cap \mathcal{S}(m_j)$). If $\mathcal{A}^{i,j}$ is not empty, then there is a constraint between v_i and v_j with the cost function $R_{i,j} = |\mathcal{A}^{i,j}|$ if $v_i = v_j, R_{i,j} = 0$ otherwise. Using these constraints, the total cost of a solution will correspond to the total number of meetings that agents cannot attend.

For our experiments, we generate an original DCOP Θ_0 and find its optimal solution α. This original problem is modified in order to obtain Θ_1, and we are now interested in solving the LTP $(\Theta_1, \alpha, \mathcal{H}, d)$ for various values of d. For generating Θ_0, two settings are considered, one with 10 meetings and 25 agents and one with 20 meetings and 50 agents. Both settings use 4 agents per meeting and 4 time periods. Θ_1 is obtained by changing two of the attending agents of each meeting and by adding two new meetings to the problem. We ran the local search algorithms for 5s, as well as the complete search, on 20 instances of each setting for various values of d.

4.9 shows the average runtime of the algorithms and the average cost of the solution obtained with instances of 10 meetings. We can clearly see that for the complete method, the time required increases exponentially with d as more and more assignments need to be explored. Regarding the cost of the solutions found, for d = 0, both LLS struggle to find good solutions. Since variables were added to the problem, there exists several assignments with a transition cost of 0. However, because of the limit d, LLS does not have much room for exploration and struggles to find good solutions. As d increases however, LLS is less limited in its exploration and the quality of its solutions gets closer to optimality.

4.7 shows the average cost of solutions obtained on instances of 20 meetings. Because of the relative complexity of these instances, the complete search could not finish within 1 hour. We see that LLS-A^o, which starts from an optimal assignment of the DCOP Θ_1, is always able to provide a better solution than LLS-α, which starts from the solution of the previous DCOP Θ_0. This is especially true for lower values of d where solutions of LLS-A^o have around 30% less cost than solution of LLS-α. This shows that while we expect a previously optimal solution (α) to be a good starting point for our search, it is actually better to start from a solution to the current problem (A^o), even if its transition cost is very high. This can be explained by the complexity of solving problems like meeting scheduling. While LLS-α has to find a good assignment while respecting the transition cost limit, LLS-A^o already has a good assignment and has to focus mostly on reducing the transition cost, which is a far easier problem. This means that, if enough time is available, a good method can be to first find the optimal solution A^o of Θ_1 and then, if its corresponding transition cost is too high, to set a limit d and solve the corresponding LTP using LLS-A^o.

4.3.4 Conclusion

Dynamic problems are a challenging topic concerning a wide array of applications that often involve a cost when adopting a new solution. In this section, we presented the definition of a Transition-Sensitive Dynamic DCOP (TS DynDCOP) which allows to take this cost into consideration. Focusing on a reactive approach, we introduced the Limited Transition Problem (LTP) where a limit d is given on the maximum transition cost allowed. We described a few methods

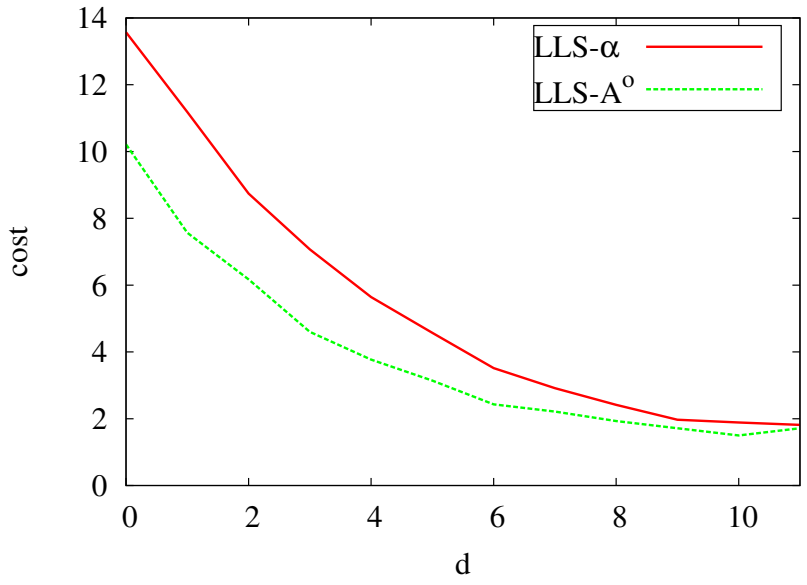


FIGURE 4.7: Solution cost when using LLS for various d (20 meetings)

to solve a LTP when the transition cost function is superadditive. For the special case where the transition cost is the Hamming distance, we were able to provide an a priori guarantee on the quality of LTP solutions.

In our experiments, we evaluated the methods presented in this section on dynamic meeting scheduling instances. We showed that it is possible to solve the LTP on simple instances using a complete approach. For more complex problems, we provided the Limited Local Search algorithm which offers a good approximation method.

As future works, we plan on exploring an approach similar to the LTP, but where instead of limiting the transition cost, we provide a minimum level of quality that should be met while minimizing the transition cost. This would offer another way to reduce the TS DynDCOP to a mono-objective problem, but we also plan to study multi-objective approaches where the whole Pareto front of trade-offs between utility and transition cost must be found. Finally, we plan on tackling real problem instances such as timetabling [96] and nurse rerostering problems [97].

Chapter 5

Related Works and Comparaison

In this chapter, we present and discuss about related works that share similar goals to this thesis. In section 5.1, we first present the previous works that aim at finding the Pareto front of multi-objective constrained problems. In section 5.2, we then present the previous works that studied ways to select a single solution or a subset from the Pareto front.

Single-Objective	Multi-Objective
Bounded Max-Sum [98]	B-MOMS [12]
ADOPT [23]	MO-ADOPT [30]
DPOP [55]	**MO-DPOP**

TABLE 5.1: Distributed complete algorithms for multi-objective constraint optimization

5.1 Algorithms for Multi-Objective Distributed Constraint Optimization Problems

5.1.1 Complete Algorithms

Complete multi-objective algorithms are ideal in the sense that they provide the full Pareto front of problems, guaranteeing to offer all trade-offs available. Due to the complexity of multi-objective problems, few complete algorithms have actually been proposed so far.

MO-DCOP

For distributed systems, MO-ADOPT [30] is the only other existing complete algorithm beside MO-DPOP (presented in section 3.1). MO-ADOPT extends ADOPT [23] , a popular search algorithm for mono-objective DCOP. MO-ADOPT guarantees to find the Pareto front of the problem by using an asynchronous search, using thresholds to prune parts of the search space and backtracking strategies to increase efficiency.

In comparison, our proposed complete algorithm MO-DPOP is an inference algorithm using dynamic programming techniques. Table 5.1 shows how MO-ADOPT is the extension of ADOPT while MO-DPOP can be seen as the extension of MO-DPOP. The main advantage of MO-ADOPT is its asynchronicity, allowing agents to constantly operates toward finding the Pareto front, maximizing their computational power. In MO-DPOP, agents have to wait to receive messages from their children to perform computations, and once an agent sent its own message, it no longer takes part in actual solving of the problem and it just waits. However, even with this difference, MO-DPOP proves to be significantly more efficient than MO-ADOPT. In the mono-objective case, ADOPT is also slower than DPOP, but each agent's memory complexity is linear while the memory complexity of DPOP is exponential. In the multi-objective case, MO-ADOPT do not possess a clear advantage with regards to the memory complexity. This is due to the set of solutions searched by MO-DCOP algorithms, i.e., the Pareto front, being exponential in the size of the problem, leading to all complete MO-DCOP algorithms to be memory exponential in the worst case.

	Single-Objective	Multi-Objective
Centralized	Bucket-Elimination [99]	Multi-Objective Bucket-Elimination [29]
Distributed	DPOP [55]	**MO-DPOP**

TABLE 5.2: Dynamic programming algorithms for multi-objective constraint optimization

MO-COP

In the centralized case, several methods have been proposed to solve Multi-Objective Constraint Optimization Problems (MO-COP). Most notably, the Multi-Objective Bucket-Elimination [29] is a complete dynamic programming approach similar to the one used by MO-DPOP. It is the extension of the Bucket-Elimination algorithm [99] that was itself extended to the distributed case with the original DPOP algorithm [55]. Table 5.2 shows the different dynamic programming-based algorithms for multi-objective problems. MO-DPOP can bee seen as the extension of the Multi-Objective Bucket-Elimination for distributed systems where a set of agents must communicate to solve the problem, introducing the need to define messages between agents and consider the costs of communication.

Another complete approach for MO-COP is the Multi-Objective AND/OR Branch-and-Bound (MO-AOBB) [11]. This algorithm uses a branch-and-bound search making use of AND/OR search trees which were previously used for mono-objective COP [100]. AND/OR search trees use a *pseudo-tree* structure such that the root is an OR node and each level alternates between OR nodes and AND nodes. In such a tree, AND nodes correspond to assigning values to variables. The main advantage of the AND/OR tree compared to classical OR trees is in its exploitation of independent variables belonging to different sub-problems, offering a much more efficient search.

The Multi-objective Best-First AND/OR search algorithm (MO-AOBF) [28] is an extension of the complete MO-AOBB algorithm that uses a best-first strategy to guide the exploration of the AND/OR tree in order to speedup the search. A guiding heuristic is required by the algorithm which greatly impact the performances.

5.1.2 Incomplete Algorithms

MO-DCOP

As far as we are aware, no incomplete algorithm have been proposed so far for distributed problems and our algorithm Bounded MO-DPOP, when using bounding functions using the weighted-sum, is the only such MO-DCOP algorithm. It is important to note however that it is possible to obtain a subset of the Pareto front of an MO-DCOP by running a succession of mono-objective DCOP algorithm. Basically, we can repeat multiple *preference-based* multi-objective

process, each time using different preferences, to obtain different Pareto optimal solution of our problem. For example, this has been previously done in distributed settings by running the mono-objective DPOP multiple times [86].

MO-COP

For MO-COP, multiple algorithms have been proposed that use the concept of ϵ-dominance [101], which relaxes the idea of Pareto dominance to allow one vector to ϵ-dominate vectors that are uncomparable with Pareto dominance. The resulting set of solutions is called an ϵ-covering.

The complete Multi-Objective AND/OR Branch-and-Bound (MO-AOBB) presented earlier was extended with depth-first heuristics [102] to provide an ϵ-covering of the Pareto front. The heuristics used were based on the multi-objective mini-bucket elimination we previously described.

5.1.3 Approximation Algorithms

Since many multi-objective optimization problems are too complex to solve with complete approaches, approximation algorithms that provide an approximation of the Pareto front are often required.

MO-DCOP

For MO-DCOPs, the Bounded Multi-Objective Max-Sum (B-MOMS) algorithm [12] is the first and only existing approximation algorithm. It is an extension of the Bounded Max-Sum algorithm [98] previously designed for mono-objective DCOPs. B-MOMS works on a factor graph where constraints and variables are nodes. Max-Sum is a dynamic programming algorithm that can find the optimal solution of a DCOP if its factor graph is a tree (no cycle). In the Bounded Max-Sum, a bounding phase is used to remove the least important edges of the graph and obtain a factor graph without any cycle. Based on the edges removed, an approximation ratio is provided to guarantee the quality of the solutions obtained. In B-MOMS, both the bounding phase and the Max-Sum phase are adapted to solve MO-DCOPs, offering a complete algorithm on cycle-less graphs, and providing a posteriori quality guarantee otherwise. B-MOMS was recently used to solve the real problem of managing water resources systems [42]

Compared to B-MOMS, our proposed approximation algorithm (DPLS) does not provide guarantees on the quality of its solutions. However, B-MOMS is requires to wait the end of the algorithm before any solution can be retrieved whereas DPLS maintains a set of solution that it

constantly attempt to improve and can thus be stopped at anytime to retrieve a solution of the problem.

MO-COP

For MO-COP, various approximation algorithms have been developed, most of them extending complete MO-COP algorithm. The Multi-Objective Mini-Bucket Elimination (MO-MBE) [29], is an extension of the inference-based Multi-Objective Bucket Elimination (MO-BE) which proposes to consider sub-problems with a bounded number of variables to avoid space and time complexity explosions. This leads to MO-MBE providing an approximation of the Pareto front whose quality depends on the bound provided. Such technique was already applied in the mono-objective case [103] and could also be used in MO-DPOP by bounding the size of the separators.

5.2 Multi-Objective Decision Making

With regards to the concept of resilience used in section 4.2, Schwind et al.(2013) first introduced the topic of systems resilience and defined a resilient system as a dynamic constraint-based model called SR-model [105]. They captured the notion of resilience for dynamic systems using several factors, i.e., resistance, recoverability, functionality and stabilizability. In our work with resilience, we focus on two properties, namely resistance and functionality, which are properties of interest underlying the resilience for DMO-COPs. Both properties are related to an important concept underlying resilience. Indeed, these properties are faithful with the initial definition of resilience proposed by Holling [81], as to ”determine the persistence of relationships within a system” and is a measure of the ability of these systems to absorb changes of state variables, driving variables, and parameters, and still persist. Bruneau’s [82] definition of seismic resilience for disaster prevention elaborates the concept of the resilience by introducing quantitative measures, which compute the triangular area of the degradation of the functionality of the system over time. In contrast, Bruneau’s [82] definition of resilience corresponds to the minimization of a triangular area representing the degradation of a system over time. This definition has been formalized under the name ”recoverability” for Dynamic COP in [105].

With regards to the concept of transition costs used in section 4.3, several works have considered transition costs in the context of *satisfaction problems*. For Dynamic Constraint Satisfaction Problems (CSP), the Minimal Perturbation Problem (MPP) [106–109] represents the problem of finding a solution with minimal transition cost from an initial assignment. The Limited Transition Problem (LTP) proposed in Section 4.3 can be seen as an extension of the MPP, where we consider a DCOP instead of a CSP and where we limit the transition cost instead of minimizing it. In a work for Dynamic SAT [110], a different approach is proposed where the sequence of

problems is assumed to be completely known in advance. This allows the use of a proactive approach to compute a sequence of solutions that minimizes the sum of transition costs over the whole sequence. Such approach could be applied to a TS DynDCOP where the sequence is known in advance. However, with the LTP, we consider a reactive case where we do not know the future problems in the sequence. For optimization problems, some of the existing works for Dynamic DCOP consider transition costs by directly incorporating them into the utility function of the problem [89, 90, 96, 111]. Compared to our approach, these works do not consider transition costs separately from the utility, using aggregation techniques to combine the two functions. Such techniques allow to remain in the domain of single-objective optimization at the cost of loosing control over the individual criteria. In comparison, considering transition costs in our proposed model turns Dynamic DCOP into a multi-objective problem. And while the LTP only has a single criterion to optimize, it still offers control over the transition cost.

With regards to our solution selection for multi-objective timetabling, while no other work considered a guaranteed subset of the Pareto front, some previous works have used criteria to directly select a single solution. The first work proposes a multi-objective evolutionary algorithm for university class timetabling [112]. By optimizing two objective functions, the authors show that better results can be obtained compared to mono-objective optimization. Such approach produces trade-off solutions between different objective functions, but with no guarantee of optimality. Compared to our work, all the solutions we provide are Pareto-optimal (and also optimal for the usual weighted-sum). The second work apply multi-objective methods to university timetabling [80] by comparing the weighted-sum to a reference point based approach, using local search algorithms. Using a reference point allows to search for a timetable as close as possible to an ideal solution. This work shows that the weighted-sum offers very unbalanced vectors and that giving a higher weight to one objective does not guarantee it will be better optimized. Compared to the weighted-sum, the reference point allows much more balanced vectors in practice but with no guarantee of obtaining the most egalitarian solution. The third work considers fairness for CB-CTT [78]. While it does not treat each soft-constraint as a single objective, it considers trade-offs between the quality of the timetable (weighted-sum) and its fairness. The fairness is defined between different university department and the goal is to spread the constraint violations equally between the different departments. They first compute a timetable with the best quality possible, and then search for interesting trade-offs with the fairness, with fairer solutions usually being of lesser quality. The final work uses Integer Programming to perform a lexicographic optimization of the objectives [113]. They consider each soft-constraint independently and optimize them one by one. The first solution obtained minimizes the first objective. Then, a second solution minimizes the second objective without increasing the first one. This is repeated for all objectives. This approach is quite effective to find a good timetable, but there exist a bias toward the first objectives optimized as there does not always exist another solution that improve the next objective without increasing the previous

ones. These works use some multi-objective techniques but either do not offer a set of solutions or cannot guarantee their optimality. Our approach has the advantage to comply with the most used solution criteria for CB-CTT (minimization of the weighted-sum) while offering more than one solution to choose from.

Chapter 6

Conclusions and Future Work

This thesis studied the process of *solving* multi-objective constraint optimization problems. The first part of this process consists in finding the Pareto front of the given problem, i.e., the set of solutions that provide an optimal trade-off of the objectives. The second part of solving multi-objective problem consists in selecting a solution from the Pareto front. This selection process is always required in practice as only one solution can actually be implemented. This thesis contributed to both parts of the *ideal* multi-objective optimization process, first by providing algorithms that compute the Pareto front of distributed multi-objective constraint optimization problems, then by providing methods that can help select a solution from the Pareto front.

6.1 Summary of contribution

The first part of our contribution is with regards to algorithms that can compute the Pareto front of Multi-Objective Distributed Constraint Optimization Problems (MO-DCOP). The first algorithm we proposed is the *Multi-Objective Distributed Pseudo-tree Optimization Procedure*. This is a *complete* algorithm that can compute the whole Pareto front of a given MO-DCOP. It relies on dynamic programming techniques and proved to be much more efficient than the previous complete MO-DCOP algorithm. We also provided an extension which uses an adjustable parameter to bound the size of messages used by the algorithm. This allows to greatly reduce the size of messages and the memory used by the algorithm while still guaranteeing a subset of the Pareto front. This extension offers the first *incomplete* MO-DCOP algorithm.

The second algorithm we proposed is the *Distributed Pareto Local Search*. This is an *approximation* algorithm that computes a set of solutions using a local search. It has the advantage of being an anytime algorithm that can yield a set of solutions at any time. When letting it run until the end, it can provide a good approximation of the Pareto front and was much better than the previous approximation MO-DCOP algorithm on medium to high density problems.

The second part of our contribution studied methods to select a single solution or a subset of solutions from the Pareto front. The first method proposed used the concept of *resilience* in proactive dynamic MO-DCOPs. Resilience offers a set of desirable properties that should always be satisfied by a solution. The method we proposed allows to isolate solutions from the Pareto front that not only satisfy resilient properties for the current problem, but also for future problems.

The second method we proposed uses the *weighted-sum*, the aggregation technique commonly used to transform a multi-objective problem into a mono-objective problem. When solving a mono-objective problem, a single solution is found, which is satisfactory in problems with a single objective. For multi-objective problems however, many different trade-offs of the objectives can provide the same optimal weighted-sum. Thus, our method proposes to find this whole set of trade-off optimizing the weighted-sum, which is guaranteed to be a subset of the Pareto front.

The third method proposed considered an objective function unique to dynamic problems, namely the *transition cost*. This cost represents the price to pay to modify a solution after it was first implemented. We proposed to select solutions from the Pareto front that limit this cost, guaranteeing a Pareto optimal solution and also providing an interesting guarantee on the quality of the other objective.

In summary, this thesis contributed to the field of distributed solving with new and more efficient algorithms suited for multi-agent systems, and to the field of multi-objective optimization by applying existing techniques in distributed settings and by developing new techniques (message bounding for MO-DPOP). We also contributed to the field of multi-objective optimization by providing methods for the *ideal* multi-objectives optimization process where an intermediate step is added in order to supply the decision maker with a subset of the Pareto front.

6.2 Perspectives

6.2.1 MO-DCOP Algorithms

At the time of writing this thesis, solving multi-objective problems in distributed systems is a quite novel research area, with the first publication being in 2011 [12]. Different approaches can be considered to solve MO-DCOPs. For example, by extending mono-objective DCOP algorithms to the multi-objective case or by extending centralized MO-COP algorithms to the distributed case. At the time of writing this thesis, all existing algorithms for MO-DCOPs were extensions of previous DCOP algorithms, with our Distributed Pareto Local Search being the only exception as it extends an MO-COP algorithms instead. MO-ADOPT extends the popular

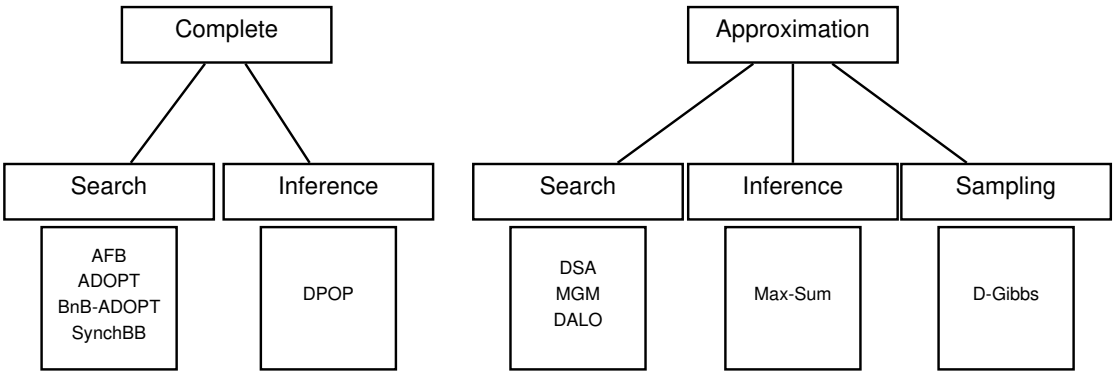


FIGURE 6.1: Taxonomy of the main DCOP algorithms.

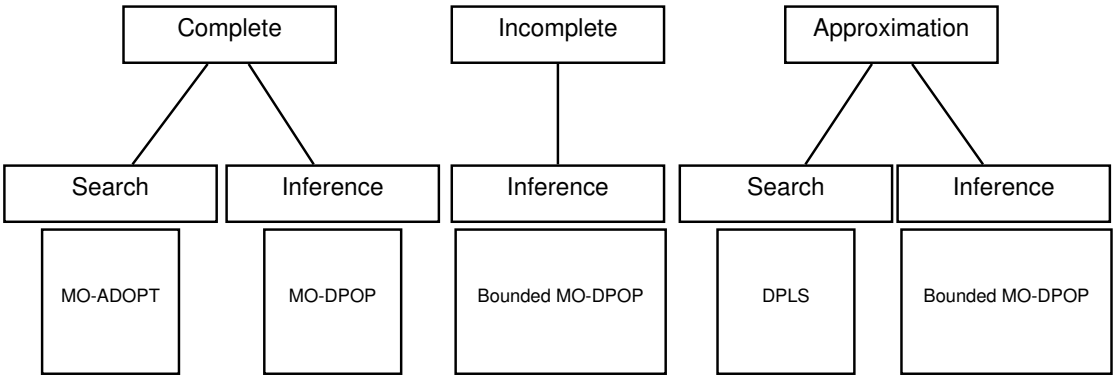


FIGURE 6.2: Taxonomy of MO-DCOP algorithms.

ADOPT algorithm, B-MOMS extends the Bounded Max Sum, and so on. However, there exists a large number of DCOP algorithms that have yet to be studied in multi-objective settings. Figure 6.1 shows some of the most popular algorithms for classical DCOPs, divided into complete algorithms that can guarantee to find an optimal solution and approximation algorithms that do not have such guarantee. In each case, algorithms can also be divided based on the techniques used, i.e., based on search, inference, or sampling techniques. Figure 6.2 shows the existing MO-DCOP algorithms, including the ones proposed in this thesis. We can make the following observation with regards to MO-DCOP algorithms:

- similarly to classical DCOP, both search and inference techniques have been studied for complete MO-DCOP algorithms (with MO-ADOPT [30] and MO-DPOP respectively);
- unlike classical DCOP, sampling [114] has not been studied for approximation MO-DCOP algorithms and should be the topic of future works;
- approximation DCOP algorithms based on search techniques (such as DSA [115], MGM [116], or DALO [117]) have not yet been extended for multiple objectives and this can be the topic of future works;
- search techniques have not yet been applied to incomplete MO-DCOP algorithms and we believe some previous search DCOP algorithms could be extended for that purpose.

The algorithms we proposed in this thesis can also be improved. The DPOP algorithm [55] and the Multi-Objective Bucket-Elimination [29], which inspired our complete algorithm MO-DPOP, have both been extended to reduce their complexity [29, 59], turning them into efficient approximation algorithm that can still provide very good solutions. Similar extensions should be considered for our own algorithm and constitute a very natural future work. Similarly, the Pareto Local Search, which inspired our approximate algorithm DPLS, has been the subject of many extensions [118]. These extensions could be applied to DPLS to improve its anytime capabilities. Moreover, extensive studies of distributed neighborhood operators should be conducted as they could provide significant problem-specific improvements.

6.2.2 Multi-Objective Applications and Benchmarks

The experimental results shown in this thesis were mostly performed on random instances, which is not always ideal. Unfortunately, there does not exist real-life instances for multi-objective constraint problems. An important next step for MO-COP and MO-DCOP would be to apply the techniques proposed in the past decade to real problems that could be then used as benchmarks to allow researchers to easily compare their works. Most applications of (mono-objective) COP and DCOP can actually be considered with multiple objectives and efforts should be made to adapt some of the existing applications and benchmarks with multiple objectives.

- Sensor networks problems [35, 36, 115, 119] can be considered with multiple objectives such as quality of observation, quality of communication between agents, total area covered, ...
- Scheduling problems [37–39] can be considered with many various constraints based on conflicts between events and preferences of users. All these different objectives can be considered independently to view the problems as MO-DCOPS.

6.2.3 Evaluating and Comparing Solution Selection Methods

There does not currently exist ways to compare different Multi-Objective Decision Making methods. The only way to evaluate a selection method is to apply it in practice Any solution selected from the Pareto front can be accepted for implementation. However, two solutions from the Pareto front can impact a system in very different ways. Thus, in the future, methods to evaluate the *impact* of selection methods should be studied. We believe that dynamic problems would be suited for such evaluation, as the impact of the selection method can be observed as the problem changes and new solutions are implemented. This is illustrated in Figure 6.3 where we can consider selecting two different solutions leading to different problems.

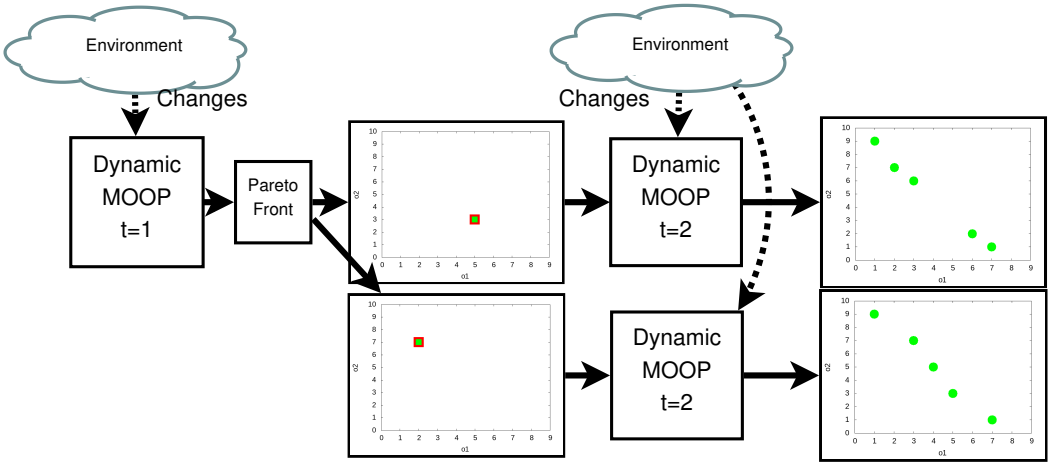


FIGURE 6.3: Impact of different selection methods on a dynamic problem.

As a first step, we proposed a simulation of dynamic mobile sensor teams [120] to observe the impact of different selection methods. In this preliminary work, *preference-based* selection methods using the weighted-sum are compared by changing the weights used. Results clearly show that the long term results obtained for the different objectives can greatly vary, even leading to undesirable extreme states. Future work should thus study additional ways to compare selection methods. In dynamic problems, self-adaptive selection methods should be studied and we believe using learning techniques could provide interesting results.

Bibliography

[1] Roger Fletcher. Practical methods of optimization. John Wiley & Sons, 2013.

[2] Christos H. Papadimitriou and Kenneth Steiglitz. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982. ISBN 0-13-152462-3.

[3] Hans Kellerer, Ulrich Pferschy, and David Pisinger. Introduction to NP-Completeness of knapsack problems. Springer, 2004.

[4] Silvano Martello and Paolo Toth. Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, Inc., New York, NY, USA, 1990. ISBN 0-471-92420-2.

[5] Kaisa Miettinen. Nonlinear multiobjective optimization, volume 12. Springer Science & Business Media, 2012.

[6] Kalyanmoy Deb and Kalyanmoy Deb. Multi-objective Optimization, pages 403–449. Springer US, Boston, MA, 2014. ISBN 978-1-4614-6940-7. doi: 10.1007/978-1-4614-6940-7_15. URL http://dx.doi.org/10.1007/978-1-4614-6940-7_15.

[7] Francesca Rossi, Peter Van Beek, and Toby Walsh. Handbook of constraint programming. Elsevier, 2006.

[8] Alan K Mackworth. Constraint satisfaction problems. Encyclopedia of AI, 285:293, 1992.

[9] Javier Larrosa and Thomas Schiex. Solving weighted csp by maintaining arc consistency. Artificial Intelligence, 159(1-2):1–26, 2004.

[10] G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search for solving constraint optimization problems. In AAAI/IAAI, Vol. 1, pages 181–187, 1996.

[11] Radu Marinescu. Exploiting problem decomposition in multi-objective constraint optimization. In Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, pages 592–607, 2009.

[12] Francisco M. Delle Fave, Ruben Stranders, Alex Rogers, and Nicholas R. Jennings. Bounded decentralised coordination over multiple objectives. In Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, pages 371–378, 2011.

[13] Jacques Ferber. Multi-agent systems: an introduction to distributed artificial intelligence, volume 1. Addison-Wesley Reading, 1999.

[14] Adelinde M Uhrmacher and Danny Weyns. Multi-Agent systems: Simulation and applications. CRC press, 2009.

[15] R Keith Sawyer. Artificial societies: Multiagent systems and the micro-macro link in sociological theory. Sociological methods & research, 31(3):325–363, 2003.

[16] Dawn C Parker, Steven M Manson, Marco A Janssen, Matthew J Hoffmann, and Peter Deadman. Multi-agent systems for the simulation of land-use and land-cover change: A review. Annals of the association of American Geographers, 93(2):314–337, 2003.

[17] Ronald C Arkin. Cooperation without communication: Multiagent schema-based robot navigation. Journal of Field Robotics, 9(3):351–364, 1992.

[18] William Yeoh and Makoto Yokoo. Distributed problem solving. AI Magazine, 33(3): 53–65, 2012.

[19] Makoto Yokoo, Edmund H Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. IEEE Transactions on knowledge and data engineering, 10(5):673–685, 1998.

[20] George E Monahan. State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms. Management Science, 28(1):1–16, 1982.

[21] Daniel S Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of markov decision processes. In Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence, pages 32–37. Morgan Kaufmann Publishers Inc., 2000.

[22] Amnon Meisels. Constraints optimization problems-COPs. Distributed Search by Constrained Agents, pages 19–26, 2008.

[23] Pragnesh J. Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. ADOPT: asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence, 161(1-2):149–180, 2005.

[24] W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. Journal of Artificial Intelligence Research, 38:85–133, 2010.

[25] R. Dechter. Constraint Processing. Morgan Kaufmann Publishers, 2003.

[26] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, pages 631–639, 1995.

[27] U. Junker. Preference-based inconsistency proving: When the failure of the best is sufficient. In ECAI, pages 118–122, 2006.

[28] Radu Marinescu. Best-first vs. depth-first and/or search for multi-objective constraint optimization. In Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence, pages 439–446, 2010.

[29] Emma Rollón and Javier Larrosa. Bucket elimination for multiobjective optimization problems. Journal of Heuristics, 12(4-5):307–328, 2006.

[30] Toshihiro Matsui, Marius Silaghi, Katsutoshi Hirayama, Makoto Yokoo, and Hiroshi Matsuo. Distributed search method with bounded cost vectors on multiple objective DCOPs. In Proceedings of the 15th International Conference on Principles and Practice of Multi-Agent Systems, pages 137–152, 2012.

[31] T. Okimoto, T. Ribeiro, M. Clement, and K. Inoue. Modeling and algorithm for dynamic multi-objective weighted constraint satisfaction problem. In Proceedings of the 6th International Conference on Agents and Artificial Intelligence (ICAART 2014), pages 420–427, 2014.

[32] Jianer Chen, Iyad A Kanj, and Weijia Jia. Vertex cover: further observations and further improvements. Journal of Algorithms, 41(2):280–301, 2001.

[33] Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In AAAI, volume 93, pages 256–262, 1993.

[34] Pierre Hansen and Brigitte Jaumard. Algorithms for the maximum satisfiability problem. Computing, 44(4):279–303, 1990.

[35] Victor Lesser, Charles L. Ortiz, and Milind Tambe, editors. Distributed Sensor Networks: A Multiagent Perspective. Kluwer Academic Publishers, 2003.

[36] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artificial Intelligence, 161(1):55–87, 2005.

[37] Amnon Meisels and Eliezer Kaplansky. Scheduling agents–distributed timetabling problems. In Practice and Theory of Automated Timetabling IV, pages 166–177. Springer, 2003.

[38] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking DCOP to the real world: efficient complete solutions for distributed multi-event scheduling. In Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems, pages 310–317, 2004.

[39] M.J. Soomer and G.J. Franx. Scheduling aircraft landings using airlines' preferences. European Journal of Operational Research, 190(1):277 – 291, 2008. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2007.06.017. URL http://www.sciencedirect.com/science/article/pii/S0377221707005693.

[40] Suguru Ueda, Atsushi Iwasaki, Makoto Yokoo, Marius C Silaghi, Katsutoshi Hirayama, and Toshihiro Matsui. Coalition structure generation based on distributed constraint optimization. In AAAI Conference on Artificial Intelligence, 2010.

[41] Robert Junges and Ana L. C. Bazzan. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems, pages 599–606, 2008.

[42] Francesco Amigoni, Andrea Castelletti, and Matteo Giuliani. Modeling the management of water resources systems using multi-objective DCOPs. In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, pages 821–829. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[43] Andrea Bettinelli, Valentina Cacchiani, Roberto Roberti, and Paolo Toth. An overview of curriculum-based course timetabling. TOP, 23(2):313–349, 2015. ISSN 1134-5764. doi: 10.1007/s11750-015-0366-z. URL http://dx.doi.org/10.1007/s11750-015-0366-z.

[44] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. Parkes, L. Gaspero, Q. Rong, and B. Edmund. Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal on Computing, 22(1): 120–130, 2010.

[45] Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. In Practice and theory of automated timetabling IV, pages 262–275. Springer, 2002.

[46] Luca Di Gaspero, Barry McCollum, and Andrea Schaerf. The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Technical report, Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1. 0, Queen's University, Belfast, United Kingdom, 2007.

[47] Alex Bonutti, Fabio De Cesco, Luca Di Gaspero, and Andrea Schaerf. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. Annals of Operations Research, 194(1):59–70, 2012.

[48] M. Banbara, T. Soh, N. Tamura, K. Inoue, and T. Schaub. Answer set programming as a modeling language for course timetabling. TPLP, 13(4-5):783–798, 2013.

[49] Tommy R Jensen and Bjarne Toft. Graph coloring problems, volume 39. John Wiley & Sons, 2011.

[50] Frank Thomson Leighton. A graph coloring algorithm for large scheduling problems. Journal of research of the national bureau of standards, 84(6):489–506, 1979.

[51] Maxime Clement, Tenda Okimoto, and Katsumi Inoue. Distributed pareto local search for multi-objective DCOPs. The Journal of the Institute of Electronics, Information and Communication: Special issue on Frontiers in Agent-based Technology, 2017. to appear.

[52] Maxime Clement, Tenda Okimoto, Katsumi Inoue, and Mutsunori Banbara. $\sum_x$-optimal solutions in highly symmetric multi-objective timetabling problems. In Proceedings of the 11th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2016), pages 63–79, 2016.

[53] Maxime Clement, Tenda Okimoto, Nicolas Schwind, and Katsumi Inoue. Finding resilient solutions for dynamic multi-objective constraint optimization problems. In Proceedings of the 7th International Conference on Agents and Artificial Intelligence (ICAART 2015), volume 2, pages 509–516, 2015.

[54] Maxime Clement, Tenda Okimoto, and Katsumi Inoue. Limiting perturbations in dynamic dcop. In Proceedings of the 30th Annual Conference of the Japanese Society for Artificial Intelligence (JSAI 2016), 2016.

[55] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In Proceedings of the 19th International Joint Conference on Artificial Intelligence, pages 266–271, 2005.

[56] N. Schwind, T. Okimoto, S. Konieczny, M. Wack, and K. Inoue. Utilitarian and egalitarian solutions for multi-objective constraint optimization. In Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on, pages 170–177, Nov 2014. doi: 10.1109/ICTAI.2014.34.

[57] Peter C Fishburn. Exceptional paper—lexicographic orders, utilities and decision rules: A survey. Management science, 20(11):1442–1471, 1974.

[58] Kalyanmoy Deb and J Sundar. Reference point based multi-objective optimization using evolutionary algorithms. In Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 635–642. ACM, 2006.

[59] Adrian Petcu and Boi Faltings. MB-DPOP: A new memory-bounded algorithm for distributed optimization. In Proceedings of the 20th International Joint Conference on Artificial Intelligence, pages 1452–1457, 2007.

[60] Tenda Okimoto, Yongjoon Joe, Atsushi Iwasaki, and Makoto Yokoo. Pseudo-tree-based algorithm for approximate distributed constraint optimization with quality bounds. In Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, pages 1269–1270, 2011.

[61] Nic Wilson, Abdul Razak, and Radu Marinescu. Computing possibly optimal solutions for multi-objective constraint optimisation with tradeoffs. In Proceedings of the 24th International Joint Conference on Artificial Intelligence, pages 815–821, 2015.

[62] Nicolas Schwind, Tenda Okimoto, Maxime Clement, and Katsumi Inoue. Representative solutions for multi-objective constraint optimization problems. In Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning, 2016.

[63] Luis Paquete, Marco Chiarandini, and Thomas Stützle. Pareto local optimum sets in the bi-objective traveling salesman problem: An experimental study. In Metaheuristics for Multiobjective Optimisation. Lecture Notes in Economics and Mathematical Systems, pages 177–200. Springer Verlag, 2004.

[64] Eric Angel, Evripidis Bampis, and Laurent Gourvés. Approximating the pareto curve with local search for the bicriteria tsp (1, 2) problem. Theoretical Computer Science, 310 (1):135–146, 2004.

[65] Arnaud Liefooghe, Jérémie Humeau, Salma Mesmoudi, Laetitia Jourdan, and El-Ghazali Talbi. On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. Journal of Heuristics, 18(2):317–352, 2012.

[66] Carlos M. Fonseca, Luís Paquete, and Manuel López-Ibáñez. An improved dimension-sweep algorithm for the hypervolume indicator. In Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006), pages 1157–1163. IEEE Press, Piscataway, NJ, July 2006.

[67] Mădălina M. Drugan and Dirk Thierens. Stochastic pareto local search: Pareto neighbourhood exploration and perturbation strategies. Journal of Heuristics, 18(5):727–766, 2012.

[68] Maarten Inja, Chiel Kooijman, Maarten de Waard, Diederik M. Roijers, and Shimon Whiteson. Queued pareto local search for multi-objective optimization. In Proceedings of the 13th International Conference on Parallel Problem Solving from Nature, pages 589–599, 2014.

[69] C. Gotlieb. The construction of class-teacher time-tables. In IFIP Congress, pages 73–77, 1962.

[70] S. MirHassani and F. Habibi. Solution approaches to the course timetabling problem. AI Review., 39(2):133–149, 2013.

[71] E. Burke, J. Marecek, A. Parkes, and H. Rudová. Decomposition, reformulation, and diving in university course timetabling. Computers & OR, 37(3):582–597, 2010.

[72] A. Rosa F. Melício, P. Calderia. Solving timetabling problem with simulated annealing. In Kluwer Academic Press, pages 171–178, 2000.

[73] P. Ross, E. Hart, and D. Corne. Genetic algorithms and timetabling. Advances in Evolutionary Computing, pages 755–771, 2003.

[74] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Proceedings of the Fifth International Conference and Symposium on Logic Programming, pages 1070–1080. MIT Press, 1988.

[75] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. Ann. Mathematics and Artificial Intelligence, 25(3–4):241–273, 1999.

[76] Chitta Baral. Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, 2003.

[77] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[78] Moritz Mühlenthaler and Rolf Wanka. Fairness in academic course timetabling. Annals of Operations Research, 239(1):171–188, 2014. doi: 10.1007/s10479-014-1553-2.

[79] Gerhard Brewka, James P Delgrande, Javier Romero, and Torsten Schaub. asprin: Customizing answer set preferences without a headache. In AAAI, pages 1467–1474, 2015.

[80] MartinJosef Geiger. Multi-criteria curriculum-based course timetabling-a comparison of a weighted sum and a reference point based approach. In Matthias Ehrgott, CarlosM. Fonseca, Xavier Gandibleux, Jin-Kao Hao, and Marc Sevaux, editors, Evolutionary Multi-Criterion Optimization, volume 5467 of Lecture Notes in Computer Science, pages 290–304. Springer Berlin Heidelberg, 2009.

[81] C. Holling. Resilience and stability of ecological systems. Annual Review of Ecology and Systematics, 4:1–23, 1973.

[82] M. Bruneau. A Framework to Quantitatively Assess and Enhance the Seismic Resilience of Communities. Earthquake Spectra, 19(4), 2003.

[83] B. H. Walker, C. S. Holling, S. C. Carpenter, and A.P. Kinzig. Resilience, adaptability and transformability. Ecology and Society, 9(2):5, 2004.

[84] P. H. Longstaff, N. J. Armstrong, K. Perrin, W. M. Parker, and M. A. Hidek. Building resilient communities: A preliminary framework for assessment. Homeland Security Affairs, 6(3), 2010.

[85] B. Cabon, S. Givry, L. Lobjois, T. Schiex, and J. Warners. Radio link frequency assignment. Constraints, 4(1):79–89, 1999.

[86] Tenda Okimoto, Nicholas Schwind, Maxime Clement, and Katsumi Inoue. Lp-norm based algorithm for multi-objective distributed constraint optimization. In Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems, pages 1427–1428, 2014.

[87] W. Yeoh, P. Varakantham, X. Sun, and S. Koenig. Incremental DCOP search algorithms for solving dynamic DCOPs. In AAMAS, pages 1069–1070, 2011.

[88] E. Hebrard, B. Hnich, and T. Walsh. Robust solutions for constraint satisfaction and optimization. In ECAI, pages 186–190, 2004.

[89] Koenraad Mertens, Tom Holvoet, and Yolande Berbers. The dyncoaa algorithm for dynamic constraint optimization problems. In Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems, pages 1421–1423. ACM, 2006. ISBN 1-59593-303-4. doi: 10.1145/1160633.1160898. URL http://doi.acm.org/10.1145/1160633.1160898.

[90] A. Petcu and B. Faltings. Optimal solution stability in dynamic, distributed constraint optimization. In Intelligent Agent Technology, pages 321–327, 2007.

[91] Graham Billiau, Chee Fon Chang, and Aditya Ghose. SBDO: A new robust approach to dynamic distributed constraint optimisation. In Proceedings of the 13th International Conference on Principles and Practice of Multi-Agent Systems, pages 11–26, 2010.

[92] F. Fioretto, E. Pontelli, and W. Yeoh. Distributed Constraint Optimization Problems and Applications: A Survey. ArXiv e-prints, February 2016.

[93] Jonathan P Pearce and Milind Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In Proceedings of the 20th International Joint Conference on Artificial Intelligence, pages 1446–1451, 2007.

[94] Maxime Clement, Tenda Okimoto, Tony Ribeiro, and Katsumi Inoue. Model and algorithm for dynamic multi-objective distributed optimization. In Proceedings of the 16th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA 2013), pages 413–420, 2013.

[95] Fabio Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi. Jade-a java agent development framework. In Multi-Agent Programming, pages 125–147. Springer, 2005.

[96] Hana Rudová, Tomáš Müller, and Keith Murray. Complex university course timetabling. Journal of Scheduling, 14(2):187–207, 2011. ISSN 1099-1425. doi: 10.1007/s10951-010-0171-3.

[97] Broos Maenhout and Mario Vanhoucke. An evolutionary approach for the nurse rerostering problem. Comput. Oper. Res., 38(10):1400–1411, October 2011. ISSN 0305-0548. doi: 10.1016/j.cor.2010.12.012. URL http://dx.doi.org/10.1016/j.cor.2010.12.012.

[98] Alex C. Rogers, Alessandro Farinelli, Ruben Stranders, and Nicholas R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. Artificial Intelligence, 175(2):730–759, 2011.

[99] Rina Dechter. Bucket elimination: A unifying framework for reasoning. Artificial Intelligence, 113(1):41–85, 1999.

[100] Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. Artificial Intelligence, 171(2-3):73–106, February 2007. ISSN 0004-3702. doi: 10.1016/j.artint.2006.11.003. URL http://dx.doi.org/10.1016/j.artint.2006.11.003.

[101] Patrice Perny and Olivier Spanjaard. Near admissible algorithms for multiobjective search. In Proceedings of the 18th European Conference on Artificial Intelligence, pages 490–494, 2008.

[102] Radu Marinescu. Efficient approximation algorithms for multi-objective constraint optimization. ADT, 11:150–64, 2011.

[103] Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. Journal of the ACM (JACM), 50(2):107–153, 2003.

[104] Christos H. Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In Proceedings of the 41st Annual Symposium on Foundations of Computer Science, pages 86–92, 2000.

[105] N. Schwind, T. Okimoto, K. Inoue, H. Chan, T. Ribeiro, K. Minami, and H. Maruyama. Systems resilience: a challenge problem for dynamic constraint-based agent systems. In AAMAS, pages 785–788, 2013.

[106] H. Sakkout and M. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints, 5(4):359–388, 2000. ISSN 1383-7133.

[107] Roman Barták, Tomáš Müller, and Hana Rudová. A new approach to modeling and solving minimal perturbation problems. In Recent Advances in Constraints, pages 233–249, Berlin, 2004. Springer.

[108] R. Zivan, A. Grubshtein, and A. Meisels. Hybrid search for minimal perturbation in dynamic CSPs. Constraints, 16(3):228–249, 2011.

[109] Alex Fukunaga. An improved search algorithm for min-perturbation. In Principles and Practice of Constraint Programming, pages 331–339. Springer, 2013. ISBN 978-3-642-40626-3. doi: 10.1007/978-3-642-40627-0_27. URL http://dx.doi.org/10.1007/978-3-642-40627-0_27.

[110] Daisuke Hatano and Katsutoshi Hirayama. Dynamic sat with decision change costs: Formalization and solutions. In Proceedings of the 22nd International Joint Conference on Artificial Intelligence, pages 560–565, 2011.

[111] Khoi D. Hoang, Ferdinando Fioretto, Ping Hou, Makoto Yokoo, William Yeoh, and Roie Zivan. Proactive dynamic distributed constraint optimization. In Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems, pages 597–605, 2016.

[112] Dilip Datta, Kalyanmoy Deb, and Carlos M. Fonseca. Multi-Objective Evolutionary Algorithm for University Class Timetabling Problem, pages 197–236. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[113] Antony E. Phillips, Hamish Waterer, Matthias Ehrgott, and David M. Ryan. Integer programming methods for large-scale practical classroom assignment problems. Computers & Operations Research, 53(0):42 – 53, 2015. ISSN 0305-0548.

[114] Duc Thien Nguyen, William Yeoh, and Hoong Chuin Lau. Distributed gibbs: A memory-bounded sampling-based dcop algorithm. In Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, pages 167–174. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[115] Weixiong Zhang, Zhao Xing, Guandong Wang, and Lars Wittenburg. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In AAMAS, volume 3, pages 185–192, 2003.

[116] Rajiv T Maheswaran, Jonathan P Pearce, and Milind Tambe. Distributed algorithms for dcop: A graphical-game-based approach. In ISCA PDCS, pages 432–439, 2004.

[117] Christopher Kiekintveld, Zhengyu Yin, Atul Kumar, and Milind Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, pages 133–140. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[118] Jérémie Dubois-Lacoste, Thomas Stützle, Mauro Birattari, and Manuel López-Ibáñez. Anytime Local Search for Multi-Objective Combinatorial Optimization: Design, Analysis and Automatic Configuration. PhD thesis, PhD thesis, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2014. 45, 2014.

[119] Roie Zivan, Harel Yedidsion, Steven Okamoto, Robin Glinton, and Katia Sycara. Distributed constraint optimization for teams of mobile sensing agents. Autonomous Agents and Multi-Agent Systems, 29(3):495–536, 2015.

[120] Maxime Clement, Tenda Okimoto, and Katsumi Inoue. Limiting perturbations in dynamic dcop. In Proceedings of the 31st Annual Conference of the Japanese Society for Artificial Intelligence (JSAI 2017), 2017.