Search and Coverage Path Planning

Thèse

Michael Morin

Doctorat en Informatique
Philosophiæ doctor (Ph.D.)

Québec, Canada

Résumé

Nous abordons deux problèmes différents et complémentaires : le problème du chemin couvrant (ou CPP) et le problème du chemin de recherche optimal (ou OSP). Le CPP est un défi important en robotique mobile alors que l'OSP est un classique de la théorie de la recherche. Nous effectuons d'abord une revue de littérature qui souligne leurs différences et leurs similitudes du point de vue d'une opération de recherche. Le CPP et l'OSP sont comparés par rapport aux données connues sur la position d'un objet de recherche. Ensuite, nous formalisons une généralisation du problème CPP aux détections imparfaites et distantes nommée CPPIED. Nous présentons un algorithme heuristique efficace qui utilise à la fois la programmation dynamique et une réduction au problème du voyageur de commerce (TSP). Nous appliquons l'algorithme dans le contexte des opérations de déminage sous-marin sur des cartes qui contiennent plus de 21 000 cellules. Nous poursuivons par l'étude d'un nouveau modèle de programmation par contraintes (CP) pour l'OSP pour lequel nous proposons une amélioration de la définition de la fonction objectif. Cette nouvelle définition permet un filtrage plus fort des variables de probabilité prodiguant ainsi une amélioration des performances du modèle. Nous proposons, pour l'OSP, une nouvelle heuristique nommée « détection totale » (ou TD). Les résultats expérimentaux démontrent que notre modèle, utilisé avec l'heuristique TD, est compétitif avec des algorithmes de séparation et d'évaluation (ou *branch-and-bound*) spécifiques au problème de l'OSP (l'approche CP étant plus générale). Cette dernière observation supporte notre assertion que la CP est un bon outil pour résoudre des problèmes de la théorie de la recherche. Finalement, nous proposons la contrainte de transition de Markov (Mtc) en tant que nouvel outil de modélisation pour simplifier l'implémentation de modèles basés sur les chaînes de Markov. Nous démontrons, tant empiriquement que formellement, que l'arithmétique des intervalles est insuffisante pour l'atteinte de la cohérence de bornes, c'est-à-dire, pour filtrer les variables de probabilité de cette contrainte. Or, l'arithmétique des intervalles est l'outil utilisé par les solveurs CP pour filtrer une Mtc lorsque celle-ci est décomposée en contraintes arithmétiques individuelles. Nous proposons donc un algorithme basé sur la programmation linéaire qui atteint la cohérence de bornes. Du fait que la programmation linéaire est coûteuse en temps de calcul pour un solveur CP lorsqu'utilisée à chaque noeud de l'arbre de recherche, nous proposons aussi une approche intermédiaire basée sur le problème du sac à dos fractionnel. L'utilisation des Mtcs est illustrée sur l'OSP.

Abstract

We tackle two different and complementary problems: the coverage path planning (CPP) and the optimal search path (OSP). The CPP is a main challenge in mobile robotics. The OSP is a classic from search theory. We first present a review of both problems that highlights their differences and their similarities from the point of view of search (coverage) operations. Both problems are positioned on the continuum of the a priori knowledge on the whereabouts of a search object. We then formalize an extension of the CPP we call the CPP with imperfect extended detections (CPPIED). We present a novel and powerful heuristic algorithm that uses dynamic programming and a traveling salesman (TSP) reduction. We apply the method to underwater minesweeping operations on maps with more than 21 thousand cells. We then study a novel constraint programming (CP) model to solve the OSP. We first improve on using the classical objective function found in the OSP definition. Our novel objective function, involving a single modification of the operators used to compute the probability of success of a search plan, leads to a stronger filtering of the probability variables of the model. Then, we propose a novel heuristic for the OSP: the total detection (TD) heuristic. Experiments show that our model, along with the proposed heuristic, is competitive with problem-specific branch-and-bounds supporting the claim that CP is a good technique to solve search theory problems. We finally propose the Markov transition constraint (Mtc) as a novel modeling tool in CP to simplify the implementation of models based on Markov chains. We prove, both empirically and theoretically, that interval arithmetic is insufficient to filter the probability variables of a single Mtc, i.e., to enforce bounds consistency on these variables. Interval arithmetic is the only available tool to filter an Mtc when it is decomposed into individual arithmetic constraints. We thus propose an algorithm based on linear programming which is proved to enforce bounds consistency. Since linear programming is computationally expensive to use at each node of the search tree of a CP solver, we propose an in-between solution based on a fractional knapsack filtering. The Mtc global constraint usage is illustrated on a CP model of the OSP.

Contents

List of Tables

List of Figures

À mes parents. . .

Acknowledgments

First and foremost, I would like to offer heartfelt thanks to my supervisor Mr. Claude-Guy Quimper and to my cosupervisor Mrs. Irène Abi-Zeid for their open-mindedness, their time, and their work. They allowed me to approach this research from a personal angle while providing precious comments, ideas and advices along the way. In crucial moments, Claude-Guy took a great care in reminding me the importance of creating solutions for which there are problems to solve and the importance of converging toward results. For Irène, theory and practice pair up well, but the winning triple is theory, practice and application. This vision reminded me to put things in perspective at every step of this research. I would also like to thank the members of the thesis jury, Mr. Philippe Galinier, Mr. François Laviolette, and Mr. Pascal Tesson, for their constructive comments and for our discussions on novel research avenues. I also take the opportunity to thank my coauthors for their scientific contribution to the various projects related in some ways to this thesis. I offer special thanks to Mr. Pascal Lang with whom I spent hours discussing the optimal search path problem as well as to Mr. Philippe Giguère who kindly offered to comment my work on the coverage problem. I would also like to take a moment to thank my colleagues from the CERMID center, from the PPC laboratory, from the GRAAL research group, and from the FORAC research consortium for all the fruitful discussions we had over the past few years. Finally, I would like to thank the anonymous reviewers for their precious comments on the publications related to this thesis as well as the editors who made these publications possible.

This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Fonds de recherche du Québec – Nature et technologies (FRQNT), and supported by the Department of Computer Science and Software Engineering of Université Laval and by the Department of Operations and Decision Systems of Université Laval. Computations required by some of the experiments were made on the supercomputer Colosse from Université Laval, managed by Calcul Québec and Compute Canada. The operation of this supercomputer is funded by the Canada Foundation for Innovation (CFI), Ministère de l'Économie, de l'Innovation et des Exportations du Québec (MEIE), RMGA and the Fonds de recherche du Québec – Nature et technologies (FRQNT).

I would like to express my gratitude to my family and to my friends: for your support, for all these laughs, and simply for being there.

J'aimerais tout d'abord offrir mes plus sincères remerciements à mon superviseur M. Claude-Guy Quimper ainsi qu'à ma cosuperviseure Mme Irène Abi-Zeid pour leur ouverture d'esprit, leur temps et leur travail. Claude-Guy et Irène m'ont permis d'approcher cette recherche sous un angle qui me tient à coeur tout en m'offrant de précieux commentaires, idées et conseils. C'est avec enthousiasme que Claude-Guy m'a rappelé, dans les moments cruciaux, l'importance de créer des solutions pour lesquelles il existe des problèmes à résoudre et l'importance de converger vers le résultat. Pour Irène, théorie et pratique font la paire, mais le triplet gagnant est théorie, pratique et application. Cette vision m'a rappelé de placer toute chose en perspective à chacune des étapes de cette recherche. Je remercie aussi M. Philippe Galinier, M. François Laviolette et M. Pascal Tesson, membres du jury, pour leurs commentaires constructifs ainsi que pour les avenues de recherche dont nous avons discuté. Je tiens aussi à remercier mes coauteurs pour leur apport scientifique aux différents projets en relation de près ou de loin avec cette thèse. J'offre un remerciement particulier à M. Pascal Lang avec qui j'ai passé des heures à discuter du problème du chemin de recherche optimal ainsi qu'à M. Philippe Giguère qui m'a offert ses commentaires relatifs à mes travaux sur le problème de couverture traité dans cette thèse. Je remercie aussi mes collègues du CERMID, du laboratoire PPC, du GRAAL et du FORAC pour nos discussions enrichissantes de ces dernières années. Finalement, je remercie les réviseurs anonymes pour leurs commentaires précieux sur les publications relatives à cette thèse ainsi que les éditeurs qui ont rendu ces publications possibles.

Cette recherche a été rendue possible grâce au financement du Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) et du Fonds de recherche du Québec – Nature et technologies (FRQNT) ainsi que grâce à l'appui du Département d'informatique et de génie logiciel de l'Université Laval et du Département d'opérations et systèmes de décision de l'Université Laval. Certains calculs requis pour les expérimentations ont été effectués sur le supercalculateur Colosse de l'Université Laval, sous la gouverne de Calcul Québec et Calcul Canada. L'exploitation de ce supercalculateur est financée par la Fondation canadienne pour l'innovation (FCI), le ministère de l'Économie, de l'Innovation et des Exportations du Québec (MEIE), le RMGA et le Fonds de recherche du Québec – Nature et technologies (FRQNT).

À ma famille, mes proches et mes amis, j'exprime ma gratitude pour leur soutien, pour tous ces rires et simplement pour leur présence.

Introduction

Path and trajectory planning problems have received a considerable attention in the scientific literature. A path can be defined as a sequence of waypoints [LaValle, 2006]. In specific cases, these waypoints pertain to a discretized physical environment where they could represent traversed regions. A trajectory, on the other hand, is a path where each segment is associated with information on the dynamic of the motion, e.g., the velocity and the acceleration of the object traveling on the segment [LaValle, 2006]. The formalisms used for both path and trajectory planning are many. The considered constraints and the optimized objective (if any) mainly depend on the decision maker's goal and hypotheses, i.e., on a real world problem to solve. For instance, we may need to minimize the total traveled distance or the threat exposure of a vehicle as it travels between two points, to detect a search object with a high probability along our way, to map an unknown environment or to catch an evasive target. In many cases the position of the goal, i.e., the endpoint to reach from a given starting position, is known. In other cases the endpoint depends on a specific objective, the target's position is unknown, and/or the environment is not fully deterministic or known leading to an uncertain location of the goal.

In this research we propose to tackle two seemingly very different families of path planning problems under uncertainty in the context of search, surveillance and detection tasks: the *coverage path planning* problems (CPPs) from the robotics literature [Choset, 2001] and the *optimal search path* problems (OSPs) [Trummel and Weisinger, 1986] from search theory [Stone, 2004]. Although CPPs and OSPs use different formalisms and come from two different domains, they are equally applicable to solve problems related to detection, search and surveillance. The hardest part, notwithstanding the inherent computational complexity of these problems, is for the decision maker to choose the right formalism for the right task.

The context of our study is not to be mingled with *object* or *target tracking*. In computer vision, object tracking usually implies to detect one or multiple objects in a scene, to identify them, and to follow (and/or predict) their motion [Yilmaz et al., 2006]. Tracking also has roots in multi-sensor data fusion. *Multi-sensor data fusion* deals with the combination of the information from various sources (sensors) in following a target's motion [Smith and Singh, 2006]. When considering a maneuvering target, such as in military applications, tracking can be seen as the modeling of that target's motion which could involve dealing with the uncertainty in the evolution of the target's trajectory (or path) [Li and Jilkov, 2003]. In all cases, the tracking phase, i.e., following the target after a detection, is an important aspect of the problem. We focus, in this research, on coverage and detection search problems that do involve both a target (search object) and a "search" phase which could possibly lead to a detection. However, in our context, no further tracking is required after that detection.

Coverage path planning (CPP) problems are often solved in order to plan an agent's path (or multiple agents' path) in such a way to guarantee complete coverage of an area [Mannadiar and Rekleitis, 2010]. In its simplest form, the coverage problem is a planning problem where the goal is to find a path of minimal length that guarantees to sweep (cover, see, scan) the whole area of interest. The goal achieved by covering an area differs from application to application. In cartography, for instance, achieving a complete coverage means obtaining a map of the area whereas in cleaning it means to send a robot to sweep an entire floor. Ultimately, one may wish to cover an area in order to find an intruder or a lost object in which case a full coverage is interpreted as a thorough and efficient survey of the area. The efficiency of the covering path can be quantified according to various criteria including path length and/or number of turns minimization which cost energy and time [Choset, 2001]. Most of the time, the implicit objective(s) can be seen as a minimization of the expenses incurred to guarantee the coverage of the area. In some cases, the coverage is uncertain and multiple passes might be required over an area to guarantee a minimal required coverage, e.g., [Gage, 1993, Drabovich, 2008]. This is often the case in search operations.

Search theory problems consist in allocating the available resources optimally in order to locate a missing search object (or maybe many missing objects) under constraints that are specific to the actual case studied, e.g., the allowed time, a vehicle's physical constraints, or the detection constraints of the sensor used to search the area. In search theory problems, it is not necessary to cover the whole area, but to maximize the chances of finding the search object(s). The first search theory report *Search and Screening*, published in 1946 and extended in 1980 [Koopman, 1980], laid the foundations of the discipline. Search theory, just as coverage path planning, has many contexts of application on which depend several families of problems. One of them is the family of *optimal search path problems* (OSPs) which is, in fact, fairly close to CPPs. The optimal search path of an OSP maximizes the global probability of finding the search object (e.g., survivor, crashed plane, lost vessel). In these detection search problems, the search stops after the first detection, an event for which we define a probability of detection conditional to the object's presence. Moreover, the probability distribution on the whereabouts of the object is usually known a priori. The whereabouts are taken into account during the planning of the search. This is not the case in most coverage problems. Although some authors assume a priori knowledge of the location of the target(s) in CPP variants [Nguyen and Hopkin, 2005, Acar et al., 2001, Zhang et al., 2001], the whereabouts of the object (if there is any object to search for) is usually unknown. In OSPs, the path length, or the allowed time, is a resource constrained by the definition of the problem. This contrasts with CPPs as well since the (possibly uncertain) coverage of the area must be guaranteed while minimizing expenses (a required coverage and a goal on resources).

With applications as different as cartography (coverage) and search and rescue (search theory), it is not a surprise that the two problem families are considered to be two different fields of research. Furthermore, the CPP and the OSP problems arose in different communities: robotics (engineering science) and operations research (management science). Even though the formalisms differ (e.g., conditional detection [Gage, 1993] versus occupancy grids [Elfes, 1989]), we argue that the whole point is to search an area of interest in a manner optimal to our specific application which strongly depends on our a priori knowledge, i.e., on the whereabouts. The links between coverage and detection become clear from a search operation perspective.

Contributions and Structure of the Document

A point of view we adopt in this thesis is to consider both OSPs and CPPs as complementary formalisms to search an area with different levels of uncertainty on the whereabouts of a search object and under different decision maker's goals and constraints. That is, the general objective of this thesis is to consider CPPs and OSPs from the unified point of view of searches. With this goal in mind, we first review, in Chapter 1, important concepts from optimization and modeling we use later on as solving tools. Thereby, we stay as close as possible to coverage and detection search problems by using problem examples related to these fields to illustrate the concepts.

We then propose, in Chapter 2, to position both families of problems on the uncertainty continuum of the a priori knowledge on the location of the goal (a goal we consider here as a search object) by a review of the related literature in both fields. This leads to the observation that both problem families are in fact close to each other in terms of applications related to searches. Recalling that CPPs tend to minimize expenses under guaranteed coverage constraints and that OSPs tend to aim at the converse goal of maximizing the effectiveness of the searches under resources constraints, it justifies the study of a particular problem from each family as both applies to search in different contexts and under different conditions. Later chapters aim at studying particular coverage and detection search problems from a combinatorial optimization perspective, namely the *CPP problem with imperfect and extended detections* (CPPIED) and the OSP from search theory. Some of the direct contributions made in the development of this thesis are published in the literature of their relevant field [Morin et al., 2012, 2013b, Morin and Quimper, 2014].

We formalize, in Chapter 3, an extension of the CPP problem to imperfect and extended detections initially introduced in [Drabovich, 2008]. We call this formalized extension, the CPPIED problem. Our formulation of the CPPIED is close to the one of the OSP in that it considers imperfect detections. The goal, in a CPPIED, is no longer to guarantee a complete coverage of the search environment, but to guarantee that a minimal required coverage in terms of probability of detection in each subarea of the area of interest is attainable by following the proposed path. Even though we describe the problem in the context of underwater minesweeping operations, the formalism is general enough to be adapted to other surveillance problems. We propose, to solve this problem, a novel heuristic that outperforms the existing technique suggested in [Drabovich, 2008] in terms of solution quality. Our algorithm, the *dynamic programming sweeper* (DpSweeper) algorithm, finds high quality solutions, in terms of path length, on grids with more than 21 thousand cells without requiring any fine-tuning and in very short time. The algorithm is built on two general ideas. First, we efficiently build a disconnected path to guarantee the searcher (here an underwater autonomous vehicle) to respect the constraint on the minimal required coverage of the area. Then, we connect the disconnected components to minimize the expenses (here the length of the path is a priority). Chapter 3 is based on an original work realized in collaboration with Irène Abi-Zeid, Yvan R. Petillot, and Claude-Guy Quimper. We presented the paper at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013) [Morin et al., 2013b]. The research conducted in [Morin et al., 2013b] was the subject, to date, of two internships ([Fruitet, 2013] and [Royer, 2014])[1].

On the search path planning side, we tackle, in Chapter 4, the OSP problem using combinatorial optimization techniques. We first present a novel *constraint programming* (CP) model and implement it in a CP solver. The solver then needs to read a model representing the problem instance to solve and to provide an optimal search path for that instance. The latter is not a trivial task for problems as complex as the OSP. We thus improve the formulation of the model by providing a modification of the objective function based on the searcher's indivisibility. This proved to slightly improve the performance of the solver but more is needed to efficiently tackle this problem. We thus develop, for the OSP problem and to improve the overall performance of the CP solver on that problem, the *total detection* (TD) heuristic which is based on search games from graph theory. The TD heuristic turned out to be an efficient technique to guide the CP solver towards high-quality search paths quickly. Chapter 4 is based on an original work realized in collaboration with Anika Pascale Papillon, François Laviolette, Irène Abi-Zeid, and Claude-Guy Quimper. We presented the paper at the 18th International Conference on Principles and Practice of Constraint Programming (CP 2012) [Morin et al., 2012]. We also collaborated on a research project studying novel bounding techniques for the OSP problem. To date, this project led to two publications we co-authored with Frédéric Simard, Claude-Guy Quimper, François Laviolette, and Josée Desharnais. The first publica-

[1] We do not claim the authorship on these reports. These are the original work of Armand Fruitet [Fruitet, 2013] and François Royer [Royer, 2014] respectively.

tion is a workshop paper entitled *Relaxation of the optimal search path problem with the cop and robber game* [Simard et al., 2014]. The paper presents a relaxation of the OSP problem into a pursuit game. This relaxation can be used to provide a bound on the objective value of an OSP. An extended version of the paper has been published in the Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP 2015). This research extends our work on the TD heuristic. The results it presents are not part of this thesis. We also collaborated, during the development of this thesis, to projects aimed at defining and discretizing search environments, i.e., the area likely to contain the object of the search. These projects lead to two publications: *Vers une planification multicritère dans le cadre de missions de recherche et sauvetage terrestres* realized in collaboration with Irène Abi-Zeid and Thanh Tung Nguyen [Abi-Zeid et al., 2011a], and *Search and Surveillance in Emergency Situations – A GIS based Approach to Construct Optimal Visibility Graphs* realized in collaboration with Irène Abi-Zeid, Thanh Tung Nguyen, Luc Lamontagne, and Patrick Maupin [Morin et al., 2013a]. Publications prior to the beginning of the present study, even if related to search and/or coverage problems, are not discussed but are available from their respective publishers.

Finally, in Chapter 5, we propose the *Markov transition constraint* (Mtc) as a novel global constraint in CP. A global constraint is basically a constraint on an arbitrary number of variables (see Section 1.2.1 of Chapter 1 for further and more formal details on the matter of global constraints and CP). Global constraints add to the expressive power of CP while enabling the use of specific algorithms to solve small subproblems of a larger problem, e.g., the computation of a single motion of the search object (i.e., our small problem) in an instance of the (larger) OSP problem. A key component of a global constraint is its filtering algorithm. In our context, a Mtc applies to a set of probability variables that represent the whereabouts before and after a motion of the object. These whereabouts are unknown (or uncertain) during the solving process performed by the solver on an OSP problem instance since the complete search path has yet to be determined. The whereabouts are represented by probability variables in an OSP model. For an incomplete search path, the value of some of these variables are not fixed leading to a loose estimate of the probabilities including that of finding the object during the search (our objective function). Considering that probability variables are represented by a lower bound and an upper bound on their plausible value, a filtering algorithm's goal is to enforce bounds consistency which means tightening these lower and upper bounds to the smallest possible intervals and, ultimately, to a singleton probability without discarding feasible solutions. Such a process, when performed efficiently, improves the performance of a CP solver. We prove, both empirically and theoretically, that interval arithmetic is insufficient to filter the probability variables of a single Mtc, i.e., to enforce bounds consistency on these variables. Interval arithmetic is the only available tool to filter an Mtc when it is decomposed into individual arithmetic constraints, i.e., when no specific filtering algorithm is used for the global constraint. We thus propose an algorithm based on linear programming which is proved to enforce bounds consistency. Since linear programming is computationally expensive to use at each node of the search tree of a CP solver, we propose an in-between solution based on a fractional knapsack filtering. The Mtc global constraint usage is illustrated on the OSP problem. The novel algorithm for the Mtc improves on the filtering performance of the CP solver we obtain on a model equivalent to the one presented in Chapter 4. Markov chains (processes) are a widely used modeling tool, from economics and business science (see [Hamilton, 1989]) to artificial intelligence (see [Russell and Norvig, 2013]). It thus remains that one of the important aspects of our contributions related to the Mtc derives from the generality of the constraint. Chapter 5 is based on our original work [Morin and Quimper, 2014] realized in collaboration with Claude-Guy Quimper. We presented the paper at the 11th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming (CP-AI-OR 2014).

Chapter 1

Optimization and Modeling: Background Theory

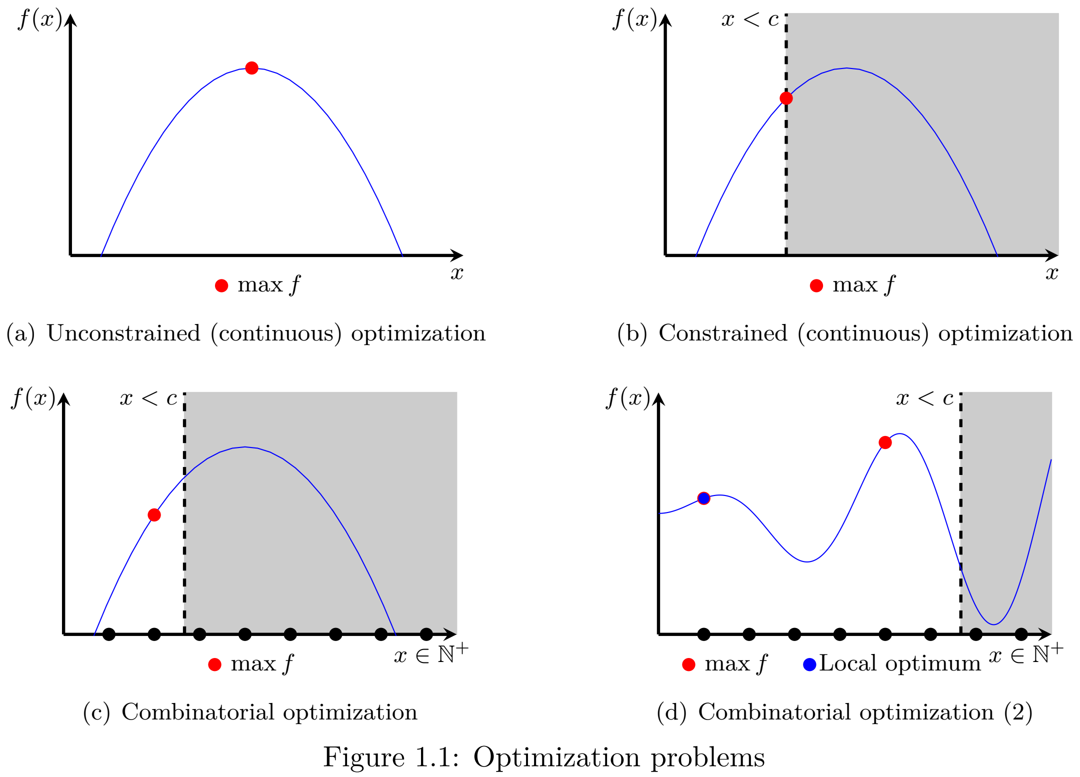
The goal of this chapter is to give the necessary background in optimization and modeling so that the tools we develop later on for coverage and detection search problems are built on a solid basis. We use, whenever possible, examples that are close to coverage and search path planning such as the traveling salesman problem (TSP), a classical coverage problem we use as part of the techniques developed in Chapter 3. The overview of the literature on coverage and search path planning is provided in Chapter 2. Here, we focus on solving and modeling techniques and we position our research with respect to mathematical programming, combinatorial optimization, heuristics and other concepts such as Markov chains and imprecise probabilities.

1.1 On Optimization and Mathematical Programming

Mathematical programming is a family of techniques for solving optimization problems. Optimization problems arise in many contexts. We mentioned, in the introduction, two well-known optimization problems: the OSP problem from search theory and the CPP problem from the robotics community. Both of them are useful in various practical contexts from search and rescue to cartography.

In a well formulated optimization problem there is a goal to fulfill, i.e., an objective function to optimize (maximize or minimize).[1] The value of that objective depends on the choices we made (e.g., the searcher's position along the path in an OSP). These decision points are called *decision variables.* When choosing the value of a variable, we *instantiate* that variable. The set of values available for a variable is the *domain* of the variable. The value of some variables may be totally dependent on other variables. Such variables are *implicit variables*

[1]We first restrict our discussion on optimization to the single objective case. An introduction to multi-objective optimization is found in Section 1.3.



(a) Unconstrained (continuous) optimization

(b) Constrained (continuous) optimization

(c) Combinatorial optimization

(d) Combinatorial optimization (2)

Figure 1.1: Optimization problems

as they do not require a direct decision to be made. Implicit variables are used for modeling purposes (e.g., the success probabilities in an OSP depend on the searcher's path and on the containment probabilities).

Optimization problems arise in different "flavors", e.g., unconstrained optimization, linear optimization, convex optimization, nonlinear optimization [Papadimitriou and Steiglitz, 1998]. The "flavor" of an optimization problem depends on the problem's practical context, i.e., the real-life unsimplified important question to answer, and, ultimately, to modeling choices made by the decision makers, the researchers and the practitioners who tackle it. Figure 1.1 presents four simple optimization problems with a different flavor.

The first one (Figure 1.1(a)) is an unconstrained problem with a single real-valued variable x. The domain of x is *continuous*. It is infinite and uncountable. The decision to be made on x is to fix its value, i.e., to instantiate it, so that the objective, $f(x)$, is maximal. In this example, a simple derivative enables us to find the maximum point of the quadratic curve (represented by the red dot on the plot).

In the second problem (Figure 1.1(b)), we add the constraint that x must be less than a constant c. A solution is said to be *feasible* if it respects the constraints on the variables. Otherwise, it is said to be an *infeasible* solution. With the additional constraint, the optimal solution of the previous unconstrained version of the problem is no longer feasible. The problem of Figure 1.1(b) is a convex problem ($f(x)$ is concave, $-f(x)$ is convex). It has a quadratic objective function and a linear inequality. This instance turns out to be simple to solve since it has a single variable. An evaluation of the graph tells us that $x \to c$ is the solution.

The third problem (Figure 1.1(c)) has a supplementary constraint on x. This constraint says that x is a natural number (zero excluded). The domain of x is still infinite, but it is countable. The choices to be made are discrete choices. Again, with only one variable, the solution is quite simple to find. However, as soon as a problem has more than one discrete domain variable each solution is a combination of choices. Some combinations lead to a maximal value of the objective function, many of them will not, and some will lead to an infeasible solution. Even if each variable has a finite domain, there is an exponential number of combinations to explore to determine the optimal objective value and thus to find an optimal solution. To further add complexity some objective functions are non-convex (Figure 1.1(d)). Even in the case of continuous variables, gradient descent do not lead to global optimality, but to a local optimum. Larger problems also have parameters. These *parameters* are constants in the definition of the problem. A *problem instance* is made of the definition of the problem (including variables and constraints) along with instantiated parameters.

We further detail these concepts in the next few sections to give a background on the tools made available for optimization purposes, especially in the combinatorial case. We effectively start, in Section 1.2, with a description of combinatorial optimization. The constraint programming (CP) approach (Section 1.2.1) is used in Chapters 4 and 5 to solve the OSP problem. Sections 1.2.2 and 1.2.3 introduce the branch-and-bound (B&B) algorithm and dynamic programming, two central techniques in combinatorial optimization and CP. These were also in the first few techniques to be tried on the OSP [Stewart, 1979, Eagle, 1984] (Chapter 2). Section 1.2.4 introduces methods to handle the complexity of large optimization problems. Along with dynamic programming (Section 1.2.3), such techniques are part of the methodology we develop in Chapter 3 to solve our coverage problem on large grids representing an ocean map. Section 1.3 summarizes important concepts for optimizing multiple contradictory objectives. The concept of lexicographic optimization (ranking of the objectives) arises in the definition of the coverage problem we tackle in Chapter 3 and in the more classical CPP problem (Chapter 2). Finally, Section 1.4 introduces Markov chains, an important modeling tool used in the OSP formalism (Chapters 2, 4, and 5). This section has three subsections. Section 1.4.1 gives a general definition of Markov chains along with an example. Section 1.4.2 reviews Markov chains with respect to the CP literature. Section 1.4.3 introduces important concepts in the theory of imprecise probability that arise in the filtering of the probability variables used in CP models with Markov chains.

1.2 Combinatorial Optimization

Combinatorial optimization can be understood as the maximization (or minimization) of an objective subject to constraints under discrete choices. Discrete choices are the essence of combinatorial optimization. Combinatorial optimization problems, in the general sense of the term, are $\mathcal{NP}$-hard[2] [Papadimitriou and Steiglitz, 1998]. Definitions 1.2.1 to 1.2.7 provide an intuitive summary of the important concepts discussed so far. We do include in the combinatorial optimization problem family problems for which implicit variables may have a continuous domain. A more formal treatment can be found in [Papadimitriou and Steiglitz, 1998].

Definition 1.2.1 (Combinatorial optimization problem)**.** A *combinatorial optimization problem* is defined by a set of decision variables representing discrete choices, a set of implicit variables, a collection of constraints, and an objective function along with parameters that are constants in the definition of the problem. $\triangleleft$

Definition 1.2.2 (Assignment)**.** A variable or a constant is *instantiated (assigned)* when its value is fixed. $\triangleleft$

Definition 1.2.3 (Combinatorial problem instance)**.** The *instance* of a combinatorial optimization problem has instantiated parameters. $\triangleleft$

Definition 1.2.4 (Candidate solution)**.** A *candidate (feasible) solution* to a combinatorial problem instance is a collection of choices instantiating the decision variables that satisfies the constraints. $\triangleleft$

Definition 1.2.5 (Feasible solutions set)**.** The *feasible solution set* $\mathcal{FSET}$ of a combinatorial optimization problem instance is the set of all candidate solutions.[3] $\triangleleft$

Definition 1.2.6 (Global optimum)**.** Given a set of feasible solutions $\mathcal{FSET}$, a solution is a *global optimum* if its objective value is at least as good as any other solution of the set. $\triangleleft$

Definition 1.2.7 (Local optimum)**.** Given a set of feasible solutions $\mathcal{FSET}$, a solution is a *local optimum* with respect to its neighborhood $\mathcal{NSET} \subseteq \mathcal{FSET}$ if its objective value is at least as good as any other solution in $\mathcal{NSET}$. $\triangleleft$

It often turns out, in combinatorial optimization, that a specific sub-structure (or sub-problem) of a larger or more complex problem can be solved efficiently. Most of the time, finding an optimal solution to an applied problem, or simply a better solution with respect

[2] Informally speaking, the $\mathcal{NP}$*-hardness* of a problem is widely recognized as a strong indication that the problem is intractable.

[3] A list of symbols is provided in appendix.

to what is usually done in practice, has an important impact on the performance of an enterprise. Sometimes, it is a matter of preserving life or of improving its quality. This is the case for problems from search theory and coverage as we can see from their applications to search and rescue and minesweeping. The next section introduces constraint programming (CP), a general combinatorial optimization solving scheme where solving sub-structures by using specific algorithms and heuristics takes an important part in efficiently solving larger and complex combinatorial problems.

1.2.1 Constraint Programming

In *constraint programming* (CP), real-world problems and/or mathematically described problems are carefully expressed in a specific language. This language is made of constraints taken from a global catalog actively developed by the CP community.[4] Translated problems are called *models*. Just as standard combinatorial optimization problems, models are made of parameters, variables, and constraints. Solvers read models along with their parameters. Once the model is read, the solver runs optimization algorithms on (a possibly translated version of) the model to instantiate its variables and find a solution. The solver runs algorithms that are specific to each model's constraint to achieve an improved performance on a global scale.

CP regroups *constraint satisfaction problems* (CSPs) and *constraint optimization problems* (COPs). CSPs are satisfaction problems whereas COPs are combinatorial optimization problems expressed in the CP language. The objective of a CSP is to find a feasible solution, i.e., to satisfy the constraints. There is, in such a problem, no objective function to optimize. We might, however, see the objective as an implicit maximization of the number of satisfied constraints. Similarly, we might see a COP as a CSP with supplementary constraints and variables that define the objective function. That is, at least one variable to represent the objective function value along with at least one constraint to maximize (or to minimize) it [van Beek, 2006, van Hoeve and Katriel, 2006]. Both CSPs and COPs are combinatorial optimization problems in that they require determining a combination of discrete choices as their solution.

Definition 1.2.8 (Domain of a CP variable)**.** The *domain* of a variable X, noted $\text{dom}(X)$ is the set of values to which the solver can instantiate that variable. ◁

Definition 1.2.9 (CSP)**.** A CSP is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ of variables, domains and constraints. The set $\mathcal{X} = \left\{X_1, \ldots, X_{|\mathcal{X}|}\right\}$ is a set of variables of which $\mathcal{D} = \left\{\text{dom}(X_1), \ldots, \text{dom}(X_{|\mathcal{X}|})\right\}$ is the set of the corresponding domains. $\mathcal{C}$ is a collection of constraints on $\mathcal{X}$. ◁

In general, the domain of a decision variable, just as in a standard combinatorial optimization problem, is countable. The values in the domain can be enumerated, in which case we have

[4] Although each CP solver has its own constraint catalog, a constraint catalog of all constraints published in the literature is maintained online [Beldiceanu and Demassey, 2014].

an *enumerated domain*, or they can be bounded, in which case we have an *interval domain* (or bounded domain).

Definition 1.2.10 (Upper bound on the domain of a CP variable)**.** The upper bound $\overline{X}$ of the domain of a CP variable X is defined as:

$$\overline{X} = \max_{x \in \mathrm{dom}(X)} x. \tag{1.1}$$

◁

Definition 1.2.11 (Lower bound on the domain of a CP variable)**.** The lower bound $\underline{X}$ of the domain of a CP variable X is defined as:

$$\underline{X} = \min_{x \in \mathrm{dom}(X)} x. \tag{1.2}$$

◁

Definition 1.2.12 (Scope)**.** The *scope* of a constraint $\mathrm{C}[X_1, \ldots, X_n]$, noted $\mathcal{SCOPE}(\mathrm{C})$, is the set of variables that are constrained by C, i.e., $\mathcal{SCOPE}(\mathrm{C}) = \{X_1, \ldots, X_n\}$. ◁

Definition 1.2.13 (Arity)**.** The *arity* of a constraint C is the cardinality of its scope, i.e., $|\mathcal{SCOPE}(\mathrm{C})|$. ◁

In CP models, simple constraints are usually expressed in mathematical language. For instance, the inequality between a variable X and a variable Y is simply expressed as $X \neq Y$. The scope of $X \neq Y$ is $\{X, Y\}$, and its arity is two. More complex constraints may either be expressed mathematically as individual constraints or as global constraints.

Definition 1.2.14 (Global constraint)**.** A *global constraint* is a relation between a variable set of non-fixed cardinality [van Hoeve and Katriel, 2006]. ◁

Example 1.2.1 (Global constraint)**.** Suppose, for instance, that we wish to express the pairwise inequality of variables X, Y, and Z. By posing $X \neq Y \wedge X \neq Z \wedge Y \neq Z$, we are not posing one constraint, but three individual constraints:

$$X \neq Y, \tag{1.3}$$
$$X \neq Z, \tag{1.4}$$
$$Y \neq Z. \tag{1.5}$$

Alternatively, the pairwise inequality of X, Y, and Z can be posed as:

$$\textsc{AllDifferent}(X, Y, Z), \tag{1.6}$$

where $\textsc{AllDifferent}(X, Y, Z)$ is a global constraint [Beldiceanu and Demassey, 2014]. ◁

One of the modeling advantage of using AllDifferent instead of a conjunction of inequalities is conciseness. The pairwise inequality of n variables is posted in only one constraint when using AllDifferent. This is a tremendous improvement in the size of the model if we compare it to the solution of posting $\binom{n}{2} = \frac{n!}{(n-2)!}$ inequality constraints, but global constraints are not only syntactic sugar. First, smaller models usually save memory. Second, by using a global constraint instead of its mathematical decomposition into individual constraints (also called *elementary constraints*), we are instructing the solver to use a specific algorithm to resolve that constraint. This modeling choice will influence the solver's performance [Régin, 1994, Smith, 2006].

Example 1.2.2 (A CSP example for the Hamiltonian cycle problem)**.** Suppose that, as a security guard, we have to visit each room of a building once during each run. We would like to avoid visiting the same room twice during the same run. Given a graph $G = (\mathcal{V}(G), \mathcal{E}(G))$ of the building where the vertices are the rooms and the edges are the corridors, our goal is to find a cycle on G that visits each vertex exactly once or to prove that such a cycle does not exist. This is the *Hamiltonian cycle problem* (HCP) which is $\mathcal{NP}$-hard [Garey and Johnson, 1979]. The graph G is a parameter of the problem. A CP model for that problem could have a set of variables representing our position at each time i of the tour. Let $TOUR_i$ (for $1 \leq i \leq |\mathcal{V}(G)|$) be such a variable. The domain of each tour variables $TOUR_i$ is the set of vertices in the graph, i.e., $\mathrm{dom}(TOUR_i) = \mathcal{V}(G)$. Since we travel between the vertices along the edges of the graph, we need the following constraints to model our movements:

$$(TOUR_i, TOUR_{i+1}) \in \mathcal{E}(G), \qquad \forall i : 1 \leq i < |\mathcal{V}(G)|. \tag{1.7}$$

The constraint $(TOUR_i, TOUR_{i+1}) \in \mathcal{E}(G)$ is an InRelation constraint more commonly known as the Table constraint [Beldiceanu and Demassey, 2014]. The pairs of values that $TOUR_i$ and $TOUR_{i+1}$ can take are explicitly defined by the set $\mathcal{E}(G)$. At this point, there is no guarantee that the last point will enable us to come back to our original position. We need to force the path to be a loop:

$$(TOUR_{|\mathcal{V}(G)|}, TOUR_1) \in \mathcal{E}(G). \tag{1.8}$$

The last constraint of the model is an AllDifferent global constraint:

$$\textsc{AllDifferent}(TOUR_1, TOUR_2, \ldots, TOUR_{|\mathcal{V}(G)|}). \tag{1.9}$$

The AllDifferent constraint forces the $TOUR_i$ variables to take distinct values. Now suppose that we are concerned by the graph of the building represented on Figure 1.2. The model is instantiated with G as a parameter. Then, the solver reads the model and provides the following solution: $TOUR_1 = a$, $TOUR_2 = b$, $TOUR_3 = c$, $TOUR_4 = d$, $TOUR_5 = e$, and $TOUR_6 = f$. ◁

Note (CP conventions). As a convention for CP models, we write constrained variables names in *ITALIC UPPER CASE*, constrained variables values in italic font and global constraints

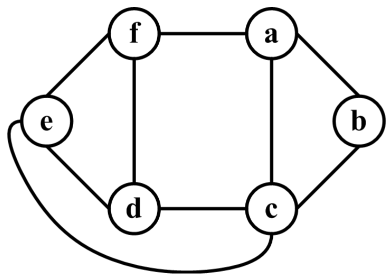


Figure 1.2: A HCP instance

names in SMALL CAPITALS. Arithmetic and "individual" constraints are expressed mathematically. To express a conjunction of constraints, we pose them on separate lines as in Example 1.2.2. ◁

As mentioned, solving a combinatorial optimization problem requires instantiating each variable to a value from its domain to obtain an optimal (feasible in the case of a CSP) solution. The space spawned by the Cartesian product of the domains is called the *decision space*. CSP solvers proceed in a *depth-first* search fashion to explore the decision space. Using a depth-first search, *partial solutions* are generated by instantiating one variable at a time.[5] Instantiating a variable X to a value $x \in \text{dom}(X)$ is as simple as posting constraint $X = x$. By doing so, the solver creates a *search tree*. The root of the search tree is an empty assignment, i.e., no variable is instantiated. The leaves are complete assignments (feasible or not). In between nodes are partial assignments. The best solution found so far is the *incumbent solution* which is, by definition, a feasible solution.

The order in which the variables are selected is a part of the solver's *search strategy* which is customizable. This is called the *variable ordering* heuristic [van Beek, 2006]. When instantiating a variable to a value from its domain, the solver is said to *branch* on that variable. The order in which the values are selected is also a part of the solver's customizable strategy. This is the *value ordering* heuristic [van Beek, 2006].

Whenever a variable is instantiated, the constraints are verified. If any constraint is violated, the solver *backtracks*, i.e., it goes up one level by removing the constraint $X = x$ posted for the instantiation. The last assigned value, which is conflictual, is removed from the domain of the variable. The solver then instantiates the variable to another value from its domain. If the removal of the conflictual value from the domain of a variable produces an empty set, i.e., $\text{dom}(X)/\{x\} = \emptyset$, the solver backtracks one more level and resets the domain. A backtrack event from the root indicates that there is no feasible solution.

Example 1.2.3 (Branching in a HCP). Suppose that we provide the HCP instance of Ex-

[5] Depending on the solver, other branching strategies might be available [van Beek, 2006]. We assume, in our discussion of CP solvers, a basic depth-first strategy with a simple backtracking system.

ample 1.2.2 to a basic CP solver. The solver reads the following instantiated model:

$$\begin{aligned}
&(TOUR_i, TOUR_{i+1}) \in \mathcal{E}(G), && \forall i : 1 \leq i < 6,\\
&(TOUR_6, TOUR_1) \in \mathcal{E}(G), && \\
&\textsc{AllDifferent}(TOUR_1, TOUR_2, \ldots, TOUR_6), && \\
&\mathrm{dom}(TOUR_i) = \mathcal{V}(G), && \forall i : 1 \leq i \leq 6,
\end{aligned} \tag{1.10}$$

where G is the graph of Figure 1.2. We assume the following static order of the variables as our variable ordering heuristic: $TOUR_1, \ldots, TOUR_6$. For our value ordering heuristic, we assume that the values are selected in the lexicographic order they appear in the domains of the variables which is $\{a, b, c, d, e, f\}$. The solver proceeds in a depth-first fashion to explore the search space. A backtrack occurs whenever a constraint is violated after an instantiation. Figure 1.3 shows the complete depth-first search tree spawned by this strategy. The leftmost branch is explored first. A straight edge is an explored edge. The path that leads to the final solution is highlighted using thick straight edges. The instantiated variables appear in blue. A dotted edge indicates an opened, but yet unexplored, node, i.e., the solver did not proceed to the instantiation of the variable. Square red nodes are nodes where a backtrack occurred. A backtrack occurred, for instance, when instantiating variable $TOUR_2$ to a since $TOUR_1$ already equals a. First, there is no loop on vertex a in the graph meaning that the assignment violates $(TOUR_1, TOUR_2) \in \mathcal{E}(G)$. Second, the assignment violates the AllDifferent constraint on the tour variables as well. Only one insatisfied constraint is necessary to trigger a backtrack. ◁

Search trees are not all created equal. Given a CP model, the usual performance metrics for solvers are both the solving time and the total number of backtracks. In Example 1.2.3, the solver backtracked 15 times using depth-first search. To improve this performance, solvers implement several methods such as forward checking and filtering algorithms [van Beek, 2006]. *Forward checking* is an algorithm that runs each time a variable X is instantiated. The algorithm iterates on all constraints C that have X in their scope and for which all variables except one, say Y, are instantiated. For each almost solved constraint C, the forward checking algorithm ensures that the instantiation of the only non-instantiated variable Y to each value $y \in \mathrm{dom}(Y)$ is consistent. That is, the assignment $Y = y$ does not violate C. If $Y = y$ violates C, it is removed from $\mathrm{dom}(Y)$. Forward checking is fast, but it only runs on constraints with one non-instantiated variable.

Filtering algorithms push the idea of forward checking a step further. A *filtering* algorithm for a constraint C prunes the values from the domains of the variables in $\mathcal{SCOPE}(\mathrm{C})$ that are inconsistent with the constraint. Just as search trees are not all created equal, there exists different levels (degrees) of consistency. A complete filtering algorithm, i.e., one that prunes all inconsistent values relative to one constraint, is said to enforce domain consistency [van Hoeve and Katriel, 2006]. That is, the domain of the variables in the scope of the constraint

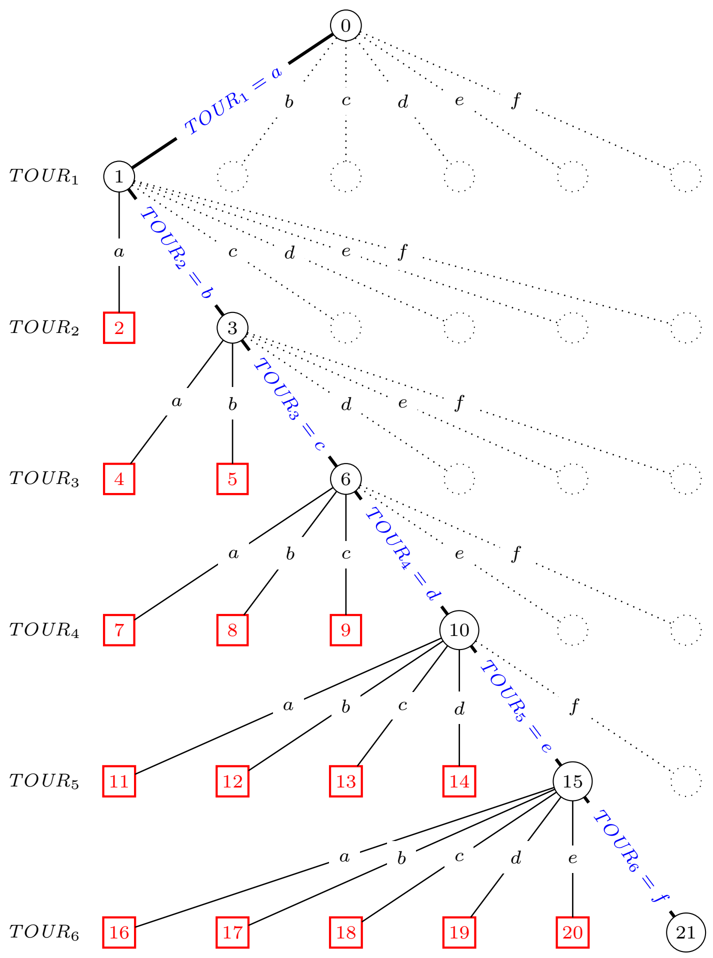


Figure 1.3: Search tree of the HCP instance of Example 1.2.3 when using a depth-first search; the nodes of the tree are numbered according to the order in which they are visited by the solver.

cannot be pruned any further without removing a solution that is feasible with respect to the constraint.

Definition 1.2.15 (Domain support)**.** Let $\mathrm{C}([X_1, \dots, X_n])$ be a constraint of arity n. The assignment $[X_1, \dots, X_n] = [x_1, \dots x_n]$ is a *domain support* if and only if $\mathrm{C}([x_1, \dots x_n])$ is satisfied and $x_i \in \mathrm{dom}(X_i)$ for all $i \in \{1, \dots, n\}$. $\triangleleft$

Definition 1.2.16 (Domain consistency)**.** A constraint $\mathrm{C}([X_1, \dots, X_n])$ is *domain consistent* if and only if, for every variable X_i, there exists a domain support with $X_i = x_i$ for all $x_i \in \mathrm{dom}(X_i)$. $\triangleleft$

Definition 1.2.17 (Interval support)**.** Let $\mathrm{C}([X_1, \dots, X_n])$ be a constraint of arity n. The assignment $[X_1, \dots, X_n] = [x_1, \dots x_n]$ is an *interval support* if and only if $\mathrm{C}([x_1, \dots x_n])$ is satisfied and the inequality $\underline{X}_i \leq x_i \leq \overline{X}_i$ holds for all $i \in \{1, \dots, n\}$. $\triangleleft$

Definition 1.2.18 (Range consistency)**.** A constraint $\mathrm{C}([X_1, \ldots, X_n])$ is *range consistent* if and only if, for every variable X_i, there exists an interval support for all $x_i \in \mathrm{dom}(X_i)$. ◁

Definition 1.2.19 (Bounds consistency)**.** A constraint $\mathrm{C}([X_1, \ldots, X_n])$ is *bounds consistent* if and only if, for every variable X_i, there exists an interval support where the variable is assigned to the lower bound of its domain $\underline{X_i}$ and a bounds support where the variable is assigned to the upper bound of its domain $\overline{X_i}$. ◁

Example 1.2.4 (A faster/better search with domain filtering in a HCP)**.** Suppose that we provide the HCP instance of Example 1.2.2 to a CP solver implementing filtering algorithms. The solver reads the instantiated model of Equation (1.10). We assume the following static order of the variables as our variable ordering heuristic: $TOUR_1, \ldots, TOUR_6$. For our value ordering heuristic, we assume that the values are selected in the lexicographic order they appear in the domains of the variables which is $\{a, b, c, d, e, f\}$. The solver proceeds in a depth-first fashion to explore the search space. Whenever a variable Y is instantiated, the solver runs filtering algorithms that are specific to each constraint C such that $Y \in \mathcal{SCOPE}(C)$. Figure 1.4 shows the complete depth-first search tree spawned by this strategy. A straight edge is an explored edge. The path that leads to the final solution is highlighted using thick straight edges. The instantiated variables appear in blue. A dotted edge indicates an opened, but yet unexplored, node, i.e., the solver did not proceed to the instantiation of the variable. There is, on this instance, no backtrack. Table B.1 of Appendix B details the solving process and the filtering that occurs at each level of the search tree. ◁

In Example 1.2.4, the solver did not backtrack. This is an improved performance in terms of total number of backtracks compared to the 15 backtracks triggered in Example 1.2.3 when using depth-first search only. This kind of performance is very rare when solving large and complex problem instances. It is a known result that the strongest level of consistency over a single constraint is domain consistency. Domain consistency is followed by range consistency and then bounds consistency. Bounds consistency and range consistency are weaker in that they prune fewer values from the domains, but they are usually faster to enforce than domain consistency. Sometimes, even enforcing bounds consistency is too expensive and weaker forms of consistency are preferred. The reason to prefer a weaker form of consistency over a stronger one is that it is often $\mathcal{NP}$-hard to enforce a stronger consistency level on a global constraint [van Hoeve and Katriel, 2006]. Choosing the filtering algorithms, and thus the desired consistency level for each constraint, is finding a trade-off between the time spent at exploring a search tree of exponential size and the time spent at running filtering algorithms to prune the domains. We now formally define the concept of consistency levels with respect to a constraint C. Let A and B be two filtering algorithms for a constraint $\mathrm{C}([X_1, \ldots, X_n])$. Let $\mathrm{dom}^{\mathrm{A}}(X_i)$ be the domain of variable X_i after running A on C. Let $\mathrm{dom}^{\mathrm{B}}(X_i)$ be the domain of variable X_i after running B on C.

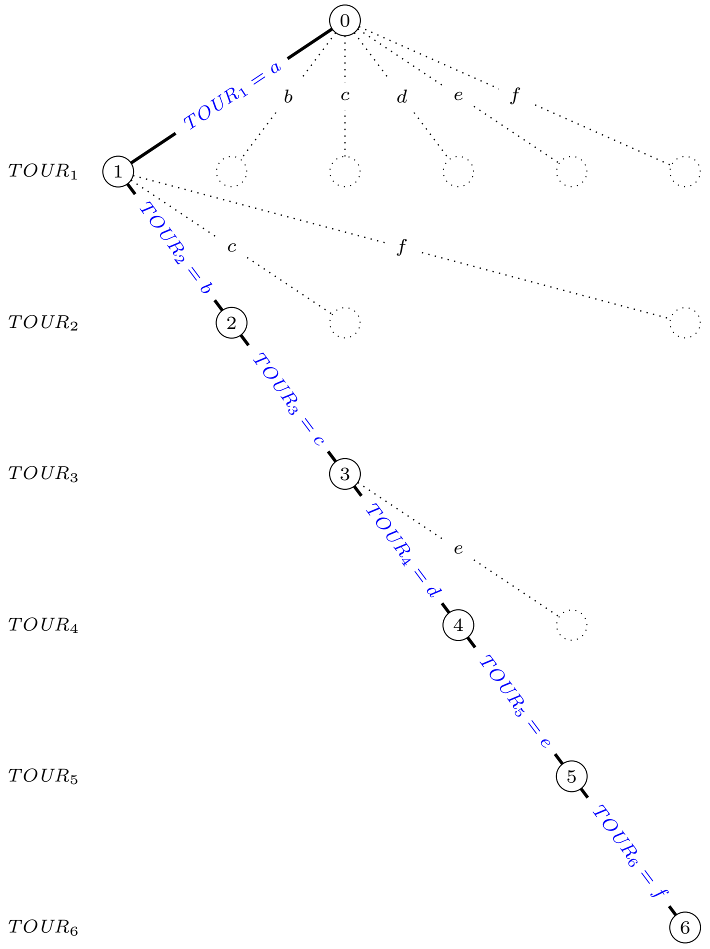


Figure 1.4: Search tree of the HCP instance of Example 1.2.4 when using a depth-first search with filtering; the nodes of the tree are numbered according to the order in which they are visited by the solver.

Definition 1.2.20 (Stronger or equal level of consistency)**.** A is said to enforce a *level of consistency* that is *stronger or equal* to the one enforced by B, noted B $\preceq$ A, if and only if $\text{dom}^{\text{A}}(X_i) \subseteq \text{dom}^{\text{B}}(X_i)$ for all variables $X_i \in \mathcal{SCOPE}(\text{C})$ for any constraint C and original domain $\text{dom}(X_i)$. ◁

Definition 1.2.21 (Strictly stronger level of consistency)**.** A is said to enforce a *level of consistency* that is *strictly stronger* than the one enforced by B, noted B $\prec$ A, if and only if B $\preceq$ A and there exist a constraint C and an original domain $\text{dom}(X_i)$ such that $\text{dom}^{\text{A}}(X_i) \neq \text{dom}^{\text{B}}(X_i)$ for at least one variable $X_i \in \mathcal{SCOPE}(\text{C})$. ◁

Theorem 1.2.1. *Let D, R, and B be three algorithms to filter a constraint $C([X_1, \ldots, X_n])$. D, R, and B respectively enforce domain consistency, range consistency, and bounds consistency. Then, it holds that $B \prec R \prec D$.*

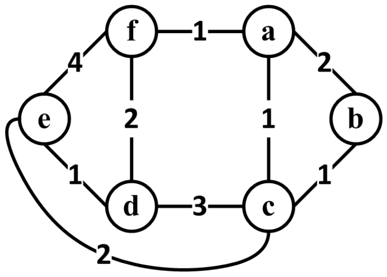


Figure 1.5: A TSP instance

Proof. The proof follows directly from the definitions. □

Up to this point, we considered discrete domain variables. The domains were either enumerated or described as an integer interval. They were, however, countable sets. There is a growing literature on continuous domain variables and constraints [Benhamou and Grandvilliers, 2006], i.e., involving real numbers.[6] Using bounded domains over real numbers is considered to be a promising approach for interval (continuous) constraints [Benhamou and Grandvilliers, 2006]. Because the domain of a variable representing a real number cannot be enumerated, various forms of bound consistency are applied to filter the domain of these variables. Still, the vast majority of solvers available online deal with countable domains only. For that reason, the discretization of continuous domain is frequent.

Optimization problems in CP are COPs. As mentioned, the only difference between CSPs and COPs is the presence, in COPs, of an objective function for which the value is represented by a variable Z. The way to handle the objective function varies from solver to solver. A common approach in CP is to solve a sequence of CSPs with constraints on the objective value variable Z [van Beek, 2006].

Example 1.2.5 (A COP example for the traveling salesman problem)**.** Suppose we refine the problem of Example 1.2.2. We still need to plan a tour to visit each room of a building represented by a graph G, but we now have further information on the travel time between each room. The travel time, in time units, is given by a function $c : \mathcal{E}(G) \to \mathbb{N}^+$. If a Hamiltonian cycle exists, we would like it to be of minimal time. This is a *traveling salesman problem* (TSP) which is, in the general case, $\mathcal{NP}$-hard [Garey and Johnson, 1979]. Suppose we need to solve the problem for the graph on Figure 1.5. The solver needs to minimize the sum of the costs of the traversed edges which we represent by a variable Z in the following

[6] Filtering bounded domain variables representing probabilities is the subject of Chapter 5 where we develop a global constraint for Markov chains.

instantiated CP model:

$$
\begin{aligned}
& Z = \min c[T_6, T_1] + \sum_{i=1}^{5} c[T_i, T_{t+1}], \\
& \text{subject to the constraints:} \\
& (\mathit{TOUR}_i, \mathit{TOUR}_{i+1}) \in \mathcal{E}(G), && \forall i : 1 \leq i < 6, \\
& (\mathit{TOUR}_6, \mathit{TOUR}_1) \in \mathcal{E}(G), \\
& \textsc{AllDifferent}(\mathit{TOUR}_1, \mathit{TOUR}_2, \ldots, \mathit{TOUR}_6), \\
& \mathrm{dom}(\mathit{TOUR}_i) = \mathcal{V}(G), && \forall i : 1 \leq i \leq 6.
\end{aligned}
\tag{1.11}
$$

The first solution found by the solver is $\mathit{TOUR}_1 = a$, $\mathit{TOUR}_2 = b$, $\mathit{TOUR}_3 = c$, $\mathit{TOUR}_4 = d$, $\mathit{TOUR}_5 = e$, $\mathit{TOUR}_6 = f$, and $Z = 12$. This is a feasible solution, but we do not know if Z is optimal. To verify its optimality, we need to solve the instance (1.11) again with the supplementary constraint that $Z < 12$. The solver finds the following solution: $\mathit{TOUR}_1 = a$, $\mathit{TOUR}_2 = b$, $\mathit{TOUR}_3 = c$, $\mathit{TOUR}_4 = e$, $\mathit{TOUR}_5 = d$, $\mathit{TOUR}_6 = f$, and $Z = 9$. To verify its optimality, we add constraint $Z < 9$. The solver finds no feasible solution. This means that 9 is the optimal value and that *abcedfa* is the optimal tour. $\triangleleft$

The approach of Example 1.2.5, although natural, seems naive as it looks like the problem needs to be solved several times from scratch. This is, however, not the case. Solvers implement it using bounds on the domain of the variable representing the objective value of the problem. Tight bounds on the objective value (either obtained by filtering algorithms during the search or by supplementary heuristics) are required to achieve good performances in the presence of a COP [van Beek, 2006]. The method we just described for COPs is also known as a *branch-and-bound* (B&B) algorithm.

1.2.2 Branch-and-Bound or the Implicit Enumeration of Solutions

The concept of search tree we discussed in the context of CSPs is a general combinatorial optimization tool that goes beyond CP. In its simplest form, a search tree is an exhaustive enumeration of all the solutions of a given problem instance. In optimization algorithms using a search tree, there is no need to perform an exhaustive enumeration of all the solutions to find the optimal one. What we need is to prove that we cannot do better on a branch than what we actually did with the current incumbent (or with the current incumbent estimate). If we can do that for a particular partial solution on a branch of the tree, we know that any solution constructed from that partial solution will not lead to an improvement of the objective value. The branch that involves that partial solution is pruned. There is no need to further explore it.

What we just described is the main idea behind B&B algorithms. B&B is not a single algorithm, but a family of algorithms that uses bounds on the objective value to prune the branches of the search tree. These algorithms perform an *implicit enumeration* of all the solutions in the tree. To do so, a B&B algorithm estimates the attainable objective value from the current partial solution. In CP, this is done by filtering. This estimate of the attainable objective value is called the bound. Suppose we are solving a maximization problem. If the bound (which is an upper bound in the context of a maximization problem) is lower than the lower bound on the objective of the incumbent solution (which is often the objective of the incumbent itself), then the partial solution is unpromising and the branch is pruned. That happens when the domain of the objective variable gets wiped-out. A bound, in the context of a maximization problem, is said to be *admissible* if it does not underestimate the objective value attainable from any partial solution. When using an admissible bound, no branch that leads to an optimal solution is pruned. The tighter a bound is, the more branches it prunes. In practice, there is a trade-off between getting a bound that is tight enough and saving computational resources.

We described the B&B procedure as done in CP although it is a general tool for mathematical programming as well. B&B algorithms are widely used in the operations research community. Especially in *integer programming* (IP) methods, shall they be mixed-integer programming methods (involving both real and integer variables) or non-linear methods (e.g., quadratic, convex) [Salkin, 1975]. In mathematical programming, they are mixed with cutting plane techniques that involves adding valid inequalities to the model to force integer solutions [Salkin, 1975]. B&B are also related to A* algorithms [LaValle, 2006] that are mainly used by the artificial intelligence community for planning. The B&B and the A* algorithms are proved to belong to the same family of methods [Labat and Pomerol, 2003]. The two algorithms describe a similar implicit enumeration of all possible solutions.

1.2.3 Dynamic Programming or Exploiting Optimal Substructures

Suppose that the combinatorial optimization problem we tackle can be divided into a sequence of recursively smaller subproblems. We say that the problem exhibits the *optimal substructure* property when the solutions to these smaller instances can be recombined to obtain the solutions to larger instances. One obvious approach, in this case, is to recursively divide the problem into its subproblems, to solve the smallest possible instances first and to recombine their solutions to obtain the solutions of the larger subproblems (see the divide-and-conquer approach [Cormen et al., 2009]). However, some of the subproblems might appear more than once in the recursion tree. Thus, some subproblems might need to be solved more than once by an algorithm. These are *overlapping subproblems*. It is often computationally more efficient to store the solution of an overlapping subproblem instead of recomputing it. This is the strategy used in *dynamic programming* [Papadimitriou and Steiglitz, 1998] for which we provide a TSP example. The dynamic programming formulation we provide as an example is attributed to [Bellman, 1962, Held and Karp, 1962].

Example 1.2.6 (A dynamic programming algorithm for the TSP)**.** Consider the graph of Figure 1.5 we used in Example 1.2.5. We assume that all pairs of vertices $u, v \in \mathcal{V}(G)$ (with $u \neq v$) such that (u, v) is not an edge of G, i.e., $(u, v) \notin \mathcal{E}(G)$, are given an infinite cost, i.e., $c(u, v) = \infty$. Furthermore, we assume that $c(u, u) = 0$. A recurrence relation for the TSP in terms of optimal substructures could be defined as follows. Let $Z_{x,\mathcal{S},z}$ be the cost of the shortest path (in terms of total cost of the traversed edges) that starts in vertex $x \in \mathcal{V}(G)$, visits all vertices of set $\mathcal{S} \subseteq \mathcal{V}(G)$, and ends in vertex $z \in \mathcal{V}(G)$. Suppose that $|\mathcal{S}| = 1$, then the minimal cost required to visit the only vertex $y \in \mathcal{S}$ before reaching vertex z is the cost of edge (x, y) plus the cost of edge (y, z). Thus, for all vertices $x, y, z \in \mathcal{V}(G)$ such that $\mathcal{S} = \{y\}$, we have that

$$Z_{x,\mathcal{S},z} = c(x, y) + c(y, z). \tag{1.12}$$

Now suppose that $|\mathcal{S}| > 1$. The minimal cost required to visit all of the vertices of set $\mathcal{S}$ exactly once is the minimum of the costs required to visit all of the vertices of a set $\mathcal{S} \setminus \{y\}$ (for $y \in \mathcal{S}$) plus the additional cost of visiting z from y:

$$Z_{x,\mathcal{S},z} = \min_{y \in \mathcal{S}} Z_{x,\mathcal{S}\setminus\{y\},y} + c(y, z). \tag{1.13}$$

Let n be the number of vertices of graph G. By computing $Z_{x,\mathcal{S}\setminus\{x\},x}$ we obtain the minimal cost of cycling over all the vertices of G from vertex x. Given x, the recurrence can be computed recursively, i.e., without storing any of the partial results. There are $(n-1)!$ tours to enumerate. This is the divide-and-conquer approach which leads to a complexity of $\Theta((n-1)!)$. When using dynamic programming, we build a table Z that stores partial solutions:

1. For all $y, z \in \mathcal{V}(G)$, initialize

$$Z_{x,\{y\},z} = c(x, y) + c(y, z). \tag{1.14}$$

2. Let $\mathcal{S}_i$ be a subset of $\mathcal{V}(G)$ of cardinality i. For each $i \in \{2, 3, \ldots, n-1\}$, subset $\mathcal{S}_i$, and $z \in \mathcal{S}_i$ updates

$$Z_{x,\mathcal{S}_i,z} = \min_{y \in \mathcal{S}_i} Z_{x,\mathcal{S}_i\setminus\{y\},y} + c(y, z). \tag{1.15}$$

The first step is done in $O(n^2)$. In the second part, we have a total of $O(2^n)$ subsets of $\mathcal{V}(G)$. For each of these subsets, we have a total of $O(n)$ choices for z, and we need to find the minimum of $O(n)$ values. This leads to an algorithm in $O(n^2 2^n)$ to compute the minimal cost of a tour on the vertices of $\mathcal{V}(G)$. The complexity of the dynamic programming algorithm is still exponential, but it is computationally more efficient than a factorial. The minimal required cost is stored at position $Z_{x,\mathcal{V}(G)\setminus\{x\},x}$. We retrieve the optimal tour by backtracking the table from $\mathcal{S} = \mathcal{V}(G) \setminus \{x\}$ while keeping track of the optimal choices:

1. Let $TOUR_1, \ldots, TOUR_n$ be the variables that will contain the optimal tour. Let $\mathcal{S}$ be the set of all vertices minus x, i.e., $\mathcal{S} = \mathcal{V}(G) \setminus \{x\}$.

2. While $\mathcal{S}$ is not empty do:

   a) Choose $y \in \mathcal{S}$ such that $Z_{x,\mathcal{S}\setminus\{y\},y}$ is minimal.

   b) Assign y to $TOUR_{|\mathcal{S}|}$, i.e., $TOUR_{|\mathcal{S}|} \leftarrow y$.

   c) Remove y from $\mathcal{S}$, i.e., $\mathcal{S} \leftarrow \mathcal{S} \setminus \{y\}$.

3. Assign x to $TOUR_1$, i.e., $TOUR_1 \leftarrow x$.

By applying this procedure on the graph of Figure 1.5 with $x = a$, we obtain $TOUR_1 = a$, $TOUR_2 = f$, $TOUR_3 = d$, $TOUR_4 = e$, $TOUR_5 = c$, $TOUR_6 = b$ which has a cost of $Z_{a,\mathcal{V}(G)\setminus\{a\},a} = 9$. $\triangleleft$

1.2.4 Further Handling Complexity

We rarely, as humans, perform an exhaustive enumeration of all the solutions to solve a problem, not even an implicit one. We use heuristics. Heuristic approaches are based on good judgment and experience [Luger, 2005]. As we discussed in the context of CP, heuristics can be used as a part of an enumerative search strategy to improve the performance of a solver. However, for complex problems and/or instances, the cost of optimality may be too high for complete algorithms to be successful in practice. This explain the large volume of scientific literature on standalone heuristics, local searches, and greedy approaches [Papadimitriou and Steiglitz, 1998].

Standalone heuristics guide the search for solutions towards "good" solutions or promising subspaces of the decision space, but they may miss global optimality [Papadimitriou and Steiglitz, 1998]. The nature of heuristics varies. We find, in the literature, methods ranging from greedy algorithms to nature inspired algorithms [Talbi, 2009]. When, we are able to obtain guarantees on the quality of a solution provided by a heuristic, we have an *approximation algorithm* [Papadimitriou and Steiglitz, 1998]. A heuristic that does locally optimal choices (see Definition 1.2.7) without looking back is a *greedy algorithm*. Depending on the problem instance to solve, greedy algorithms might lead to global optimality.

Example 1.2.7 (Greedy choices for the TSP)**.** Given a partial solution to a TSP problem on a complete graph G defined as a sequence of edges (path) ending by arc (u, v), a greedy choice would be to select x such that the cost $c(v, x)$ is minimal. A greedy algorithm implementing this heuristic would simply be to start from an arbitrary vertex s, to perform a sequence of greedy choices until the growing path has traversed all the vertices of G, and to come back to s. This algorithm is known as the *nearest neighbor algorithm* for the TSP. Suppose that the cost function c is positive and symmetric and that it satisfies the triangle inequality. Then, this simple heuristic, which involves greedy choices in the neighborhood of the last vertex added to the path, is known to produce solutions with a total length of at most $\frac{1}{2}\lceil \log_2 n \rceil + \frac{1}{2}$ times the optimal length where n is the number of vertices in the graph [Rosenkrantz et al., 1977].[7] ◁

Most algorithms we discussed up to this point are *constructive* approaches in that they start with an empty solution and iteratively complete it [Hoos and Stützle, 2004]. Suppose we already have a feasible solution to the problem at hand or that it is easy to construct one. We might try to improve that solution instead of building a new one from scratch. To do so, we modify some variable assignments to obtain a new solution. This is the concept behind *perturbative* approaches [Hoos and Stützle, 2004]. *Local searches* are often viewed as perturbative algorithms. A local search algorithm starts at a point in the decision space, i.e., from a solution. It then successively moves from neighbor to neighbor until its termination criterion is met. This process, easily seen in terms of a perturbative algorithm, can be performed in a constructive fashion [Hoos and Stützle, 2004]. Ant colony optimization [Dorigo et al., 1996] is an example of constructive local search where each ant builds a solution using a shared knowledge stored in the *pheromone trails* accumulated in the search space. Ant colony optimization algorithms often involve an additional perturbative local search step [Dorigo and Blum, 2005]. In any case, local searches need a neighborhood operator that, when applied to a solution of the decision space, returns a collection of solutions. Deterministic or stochastic neighborhood operators can be used in local searches. The latter case leads to what we call a *stochastic local search* algorithm.

A heuristic describing a general strategy in problem solving rather than problem-specific rules is called a *metaheuristic* [Talbi, 2009]. Metaheuristics are appealing in that they are often inspired by nature[8]. Their design, however, involve extensive experiments and careful adaptation to the problem at hand.

1.3 Multi-objective Optimization

A multi-objective optimization problem (combinatorial or not) involves not a single, but n objective functions. The domain of each objective f_i $(1 \leq i \leq n)$ is the decision space. We assume, for our discussion, that the image of a solution S from that decision space, i.e., $f_i(S)$, is in $\mathbb{R}$. Then, $F(S) = [f_1(S), f_2(S), \ldots, f_n(S)]$ is the objective vector associated with S.[9] The point $F(S) \in \mathbb{R}^n$, i.e., the image of S in the objective space, is its value in terms of all objectives. We suppose, without loss of generality, that all objectives are to be minimized.

[7] Other approximation algorithms are known to achieve a better approximation ratio than the nearest neighbor algorithm for this special TSP case. Christofides' algorithm, for instance, is guaranteed to return a solution with a total length of at most $\frac{3}{2}$ times the optimal length [Papadimitriou and Steiglitz, 1998].

[8] Ant colony optimization is an example of nature-inspired metaheuristic.

[9] An alternative formulation is to consider a set of objectives rather than a vector.

Even so, objectives might be conflictual or incommensurable. For that reason, the goal of multi-objective optimization is not to find the best feasible solution, but to find one that is in accordance with the decision maker's preferences. Such preferences may be articulated a priori, a posteriori or interactively. A priori preferences articulation [Miettinen, 2008] takes into account expressed preferences of the decision maker such as her/his goals and her/his aspirations prior to the solving process. Interactive preferences articulation [Miettinen et al., 2008] deals with approaches where the decision maker continuously expresses her/his preferences as the algorithm searches for solutions. A posteriori preferences articulation [Miettinen, 2008] includes methods where the algorithm produces a set of efficient (acceptable, reasonable) solutions before taking the decision maker's preferences into account. This set ultimately needs to be reduced to a single solution that reflects her/his goals and aspirations. A reasonable notion of *efficiency* is that of *Pareto optimality*. Pareto optimal solutions are feasible solutions that are also *non-dominated*.

Definition 1.3.1 (Dominance relation)**.** A feasible solution S dominates a feasible solution S', written S dominates S', if and only if

$$(\forall i : f_i(S) \leq f_i(S')) \wedge (\exists j : f_j(S) < f_j(S')) \, . \tag{1.16}$$

In other words, S is at least as good as S' on all objectives and strictly better on at least one of them. ◁

Definition 1.3.2 (Pareto optimality)**.** A *Pareto optimal* solution S is not dominated by any other feasible solution, i.e., $\neg\,(S'$ dominates $S)$ $(\forall S' \in \mathcal{FSET})$. ◁

Common techniques for solving multi-objective optimization problems are the approximation of the Pareto optimal set by metaheuristics, the aggregation of the objective functions into a single objective, and lexicographic optimization which is also known as goal programming [Talbi, 2009]. *Lexicographic optimization* involves a ranking of the objectives according to their preferred order. A common technique of lexicographic optimization is to sequentially optimize the objectives in that order. Each subsequent optimization on an objective with a lower priority includes constraints to avoid a substantial decrease in the objective value of the previously optimized objectives which are of a higher priority. A solution is said to be *lexicographically optimal* if it is non-dominated in the lexicographic sense.

Definition 1.3.3 (Lexicographic dominance)**.** Suppose the order of the objectives in F corresponds to their rank from the highest to the lowest preference. A feasible solution S lexicographically dominates a feasible solution S' with respect to an ordering of the objectives $f_1, f_2, \ldots, f_n$ if and only if there exists $1 \leq k \leq n$ such that

$$(\forall i < k : f_i(S) = f_i(S')) \wedge f_k(S) < f_k(S'). \tag{1.17}$$

In other words, S is as good as S' on the first $(k-1)^{\text{th}}$ objectives while being strictly better on the k^{th} objective. ◁

In Chapter 3, we use the concept of lexicographic dominance from Definition 1.3.3 to compare the solutions provided by heuristics on a bi-objective coverage problem. Non-dominated solutions in the lexicographic sense are Pareto optimal [Talbi, 2009].

1.4 Markov Chains

Markov processes (also called Markov chains) are central to many applications. They are used to model the motion of a target in a physical environment [Stone, 2004]. In computer science they are used in the Pagerank algorithm used by Google [Page et al., 1999]. They are also used in economics and business science [Hamilton, 1989]. They form the basis of decision making frameworks, such as Markov decision processes and hidden Markov decision processes which are fundamental to many applications in artificial intelligence [Russell and Norvig, 2013]. They even apply to arts for the generation of melodies [Pachet et al., 2011] and lyrics [Barbieri et al., 2012]. In this section, we first define Markov chains and Markov transitions (Section 1.4.1). We then provide an overview of Markov chains as a modeling tool in combinatorial optimization and more precisely in CP (Section 1.4.2). Finally, we discuss how Markov chains can be related to the theory of imprecise probability when considering them from a CP point of view (Section 1.4.3), a tool we use in Chapter 5.

1.4.1 A Definition of Markov Chains with a Motion Model Example

Let $\mathcal{N} = \{1, \dots, N\}$ be a space of N states. States are mutually exclusive and jointly exhaustive, i.e., the process is in exactly one state of $\mathcal{N}$ at any time. Let $\mathbf{M}$ be the $N \times N$ *transition matrix* of the Markov process. That is, for all $i, j \in \mathcal{N}$, $\mathbf{M}_{ij}$ is the probability of moving from state i to state j at any step. The total probability of moving from state $i \in \mathcal{N}$ (given that the process is in state i) is 1. That is,

$$\sum_{j \in \mathcal{N}} \mathbf{M}_{ij} = 1, \qquad \forall i \in \mathcal{N}. \tag{1.18}$$

Let $\mathbf{x}^t = [x_1^t, \dots, x_N^t]$ be the row vector of the probability distribution on the states at a time $t \in \{1, \dots, T\}$ where x_i^t is the probability that the process is in state $i \in \mathcal{N}$ at step $t \in \{1, \dots, T\}$. Given an initial distribution $\mathbf{x}^1$ over the states such that $\sum_{i \in \mathcal{N}} x_i^1 = 1$ and $0 \le x_i^1 \le 1$ for all $i \in \mathcal{N}$, the *Markov property* states that

$$\mathbf{x}^{t+1} = \mathbf{x}^t \mathbf{M}, \qquad \forall t \in \{2, \dots, T\}. \tag{1.19}$$

That is, the state at time $t+1$ depends on the previous state only. Some generalizations of Markov chains allow the current state to depend on d previous states where $d > 1$ (e.g., Markov chains of order d). We restrict ourselves to previous state dependencies only, i.e., to first-order Markov chains. Given the distribution $\mathbf{x}^t$, the distribution after k steps from step t, $\mathbf{x}^{t+k}$, is computed as:

$$\mathbf{x}^{t+k} = \mathbf{x}^t \mathbf{M}^k, \qquad \forall t \in \{1, \dots, T\}, k \in \{0, \dots, T-t\}. \tag{1.20}$$

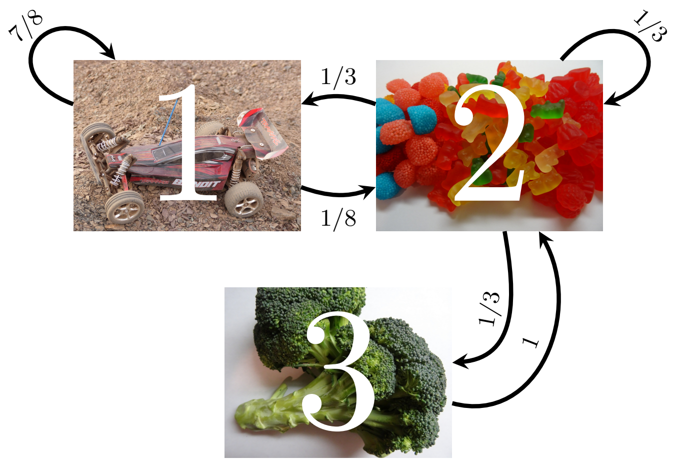


Figure 1.6: A mall with three stores and the related transition probabilities of a lost child (fictional).

Example 1.4.1. Suppose that we wish to model the motion of a lost child between three stores (store 1, 2, and 3) of a mall with discrete time intervals of one minute. The child's behavior is as follows:

- s/he may spend time in the toys store (number 1);
- after one minute, her/his probability of leaving to the candy store (number 2) is $\frac{1}{8}$;
- from the candy store, s/he either returns to the toys store, stays there, or goes to the food market (number 3);
- whenever the child is in the food market, s/he directly returns to the candy store.

Figure 1.6 shows the environment where the child is lost. The states, representing the child's plausible locations, are numbered from 1 to 3. Transition probabilities are displayed on each arcs. The resulting child's motion model is a transition matrix $\mathbf{M}$:

$$\mathbf{M} = \begin{bmatrix} \frac{7}{8} & \frac{1}{8} & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 1 & 0 \end{bmatrix}. \tag{1.21}$$

The source states (from 1 to 3) are on rows. The destination states (from 1 to 3) are on columns. We see, for instance, that the probability of a child's movement from the food market (state 3) to the candy store (state 2) is 1.

Suppose that the child starts in store 1. That is, the probability distribution at time 1 is $\mathbf{x}^1 = [1, 0, 0]$. We may, using equation (1.19), infer that the distribution over the child's location after a minute is $\mathbf{x}^2 = \left[\frac{7}{8}, \frac{1}{8}, 0\right]$. It is as easy to know, using equation (1.20), the distribution over the child's location after a delay of $k > 1$ minutes. ◁

1.4.2 Markov Constraints in the Literature

Markov chains (processes) are a widely used modeling tool. Constraint programming makes no exception as there exist, in the literature, constraints to tackle models based on Markov chains. One approach is the decomposition of the Markov chain into individual arithmetic constraints. As discussed in Chapter 5 this method is, however, not sufficient to guarantee an optimal filtering.

A second approach is the use of global constraints. Pachet and Roy [2011] introduced the *elementary Markov constraint* (Emc). The Emc is general enough to take d-order Markov chains (with $d > 1$) into account [Pachet and Roy, 2011]. For the purpose of this review, we restrict our Emc definition to first-order Markov chains.

Definition 1.4.1 (The elementary Markov constraint)**.** Let S and S' be two variables that represent states in $\mathcal{N}$. The domains of the state variables S and S' are subsets of $\mathcal{N}$:

$$S, S' \subseteq \mathcal{N}. \tag{1.22}$$

Let $P_{S'}$ be a variable representing a probability. The domain of the probability variable $P_{S'}$ is the set of conditional probabilities of achieving state S' from any previous state:

$$\text{dom}(P_{S'}) = \{p \mid \exists i \in \mathcal{N}, p = \mathbf{M}_{iS'}\}. \tag{1.23}$$

These probabilities are computed prior to the solving process during the generation of the model. Given $\mathbf{M}$, a known Markovian transition matrix, the Emc is defined as follows:

$$\textsc{Emc}(S, S', P_{S'}) \Leftrightarrow P_{S'} = \mathbf{M}_{SS'}. \tag{1.24}$$

The constraint $\textsc{Emc}(S, S', P_{S'})$ states that the probability of moving from state S to state S' is $P_{S'}$. ◁

Using multiple Emcs, the authors model *constrained Markov processes*, i.e., Markov processes with supplementary constraints on the generated sequence. Let $S_1, \ldots, S_T$ be the state variables of a sequence of a first-order Markov chain. Let $P_{S_2}, \ldots, P_{S_T}$ be the variables that represent the probabilities. The constrained sequence is modeled as chained Emcs:

$$\textsc{Emc}(S_t, S_{t+1}, P_{S_{t+1}}), \qquad \forall t \in \{1, \ldots, T-1\}. \tag{1.25}$$

The Markov property, enforced by the Emcs, is a cost function to optimize whereas supplementary constraints are used to steer the generation of the sequence. The approach is applied to the generation of melody and chord sequence.

Following Pachet and Roy [2011], Pachet et al. [2011] show that when the scope of the supplementary constraints does not exceed the order of the chain d, the constrained Markov process may be recompiled into a statistically equivalent unconstrained Markov process. The approach is illustrated on the melody generation problem. In [Barbieri et al., 2012], the authors apply constrained Markov processes to the generation of lyrics.

We introduce, in Chapter 5, the *Markov transition constraint* (Mtc). The Mtc differs from the *elementary Markov constraint* (Emc) presented in [Pachet and Roy, 2011, Pachet et al., 2011]. The Mtc keeps, at a time step $t \in \{1, \ldots, T\}$, a distributions $\mathbf{x}^t$ over the states. The Emc keeps, at a time step $t \in \{1, \ldots, T\}$, a single state. Moreover, the Mtc deals with interval-domain probabilities while the Emc deals with finite-domain probabilities each probability being computed during the generation of the model. The next section gives a broad overview of the concept of probabilities defined as intervals.

1.4.3 Imprecise Markov Chains

Imprecise (uncertain) Markov chains (e.g., [Blane and den Hertog, 2008, de Cooman et al., 2009, Škulj, 2009]) are Markov chains with *imprecise probabilities* [Coolen et al., 2011]. The probabilities of the transition matrix and of the initial distribution over the state space are *imprecise* in the sense that they are given as credal sets (sets of probability measures) which may be represented as probability intervals instead of classical (singleton) probabilities. A complete introduction to imprecise Markov chains goes far beyond the scope of this thesis since we deal, in our models, with "precise" Markov chains where both the transition matrix and the initial distribution are defined using the usual concept of probabilities. The introduction of imprecise probability concepts is nonetheless desirable due to the filtering of probability variables in CP models. When filtering the probability distributions resulting of the application of a Markov chain in CP, the probability variables are not fixed. That is, they are defined as probability intervals which are imprecise probabilities.

The theory of the imprecise probabilities is more general than the one of "classical" probabilities [Coolen et al., 2011]. Singleton probabilities are a specific case of imprecise probability theory where the interval is reduced to a single value. Just as optimization problems has many flavors, the theory of imprecise probability has. Perhaps one of the most general formulation of the theory in terms of interpretation is that of Weichselberger [2000] who builds the theory from the Kolmogorov axioms. It is not our intent to go into the details of the theory since it is often sufficient, for our filtering purposes in CP, to consider that the probability $\widehat{\Pr}(\mathsf{E})$ of an event E is imprecise if it is defined as an interval $\widehat{\Pr}(\mathsf{E}) = [\underline{\Pr}(\mathsf{E}), \overline{\Pr}(\mathsf{E})]$. This statement, however, has a lot of subtle implications that can be clarified by a summary of the axioms of the theory of imprecise probability we summarize in Definitions 1.4.2 to 1.4.4. It is also worth mentioning that the challenges involved in solving the computational problems underlying the theory (e.g., filtering an imprecise Markov chain) are still opens [Coolen et al., 2011]. Interested readers are referred to [de Campos et al., 1994, 1995, Weichselberger, 2000]

Definition 1.4.2 (The Kolmogorov axioms and the probability space)**.** Let Ω be a sample space, and $\mathcal{F}$ be a σ-field[10] of random events $\mathcal{F} \subseteq \Omega$, i.e., an event space. A function Pr with domain $\mathcal{F}$ is a *Kolmogorov-function* (or *K-function*) if it satisfies:

1. $\Pr(\mathsf{E})$ of an event $\mathsf{E} \in \mathcal{F}$ is a non-negative real number;
2. $\Pr(\Omega) = 1$;
3. any sequence of mutually exclusive events E_1, E_2, ... (with $\mathsf{E}_i \in \mathcal{F}$ for $i > 0$) satisfies:

$$\Pr(\mathsf{E}_1 \cup \mathsf{E}_2 \cup \ldots) = \sum_{i}^{\infty} \Pr(\mathsf{E}_i). \tag{1.26}$$

It follows from the three axioms that $\Pr(\emptyset) = 0$, and that $\Pr(\mathsf{E})$ is in the interval $[0, 1]$ for any event $\mathsf{E} \in \mathcal{F}$. The triplet $(\Omega, \mathcal{F}, \Pr)$ is called the probability field. $\triangleleft$

Definition 1.4.3 (Reasonable (imprecise) probability)**.** Let $\widehat{\Pr}(\mathsf{E})$ be an imprecise probability defined as an interval $[\underline{\Pr}(\mathsf{E}), \overline{\Pr}(\mathsf{E})]$. Let $\mathcal{M}$ be the set of Kolmogorov functions on $\mathcal{F}$ for which the probability of any events is within the bounds:

$$\mathcal{M} = \left\{\Pr : \left(\forall \mathsf{E} \in \mathcal{F} : \underline{\Pr}(\mathsf{E}) \leq \Pr(\mathsf{E}) \leq \overline{\Pr}(\mathsf{E})\right)\right\}. \tag{1.27}$$

$\widehat{\Pr}$ with domain $\mathcal{F}$ is a *reasonable (imprecise) probability* (or *R-probability*) if:

1. for any $\mathsf{E} \in \mathcal{F}$, $\widehat{\Pr}(\mathsf{E}) = [\underline{\Pr}(\mathsf{E}), \overline{\Pr}(\mathsf{E})]$ is such that:

$$0 \leq \underline{\Pr}(\mathsf{E}) \leq \overline{\Pr}(\mathsf{E}) \leq 1; \tag{1.28}$$

2. $\mathcal{M}$ is not empty.

It follows from the definition that $\underline{\Pr}(\emptyset) = 0$, and $\overline{\Pr}(\Omega) = 1$. The quadruplet $(\Omega, \mathcal{F}, \underline{\Pr}, \overline{\Pr})$ is called a reasonable probability field. $\triangleleft$

Definition 1.4.4 (Feasible (imprecise) probability)**.** A $\widehat{\Pr}$ with domain $\mathcal{F}$ is a *feasible (imprecise) probability* (or *F-probability*) if:

1. it is a reasonable probability;
2. for any $\mathsf{E} \in \mathcal{F}$, the lower bound on $\widehat{\Pr}(\mathsf{E})$ is attainable:

$$\inf_{\Pr \in \mathcal{M}} \Pr(\mathsf{E}) = \underline{\Pr}(\mathsf{E}); \tag{1.29}$$

[10] $\mathcal{F}$ is a σ-field (or σ-algebra) of a set Ω if the following holds: (i) $\mathcal{F} \subseteq 2^{\Omega}$, (ii) $\mathcal{F} \neq \emptyset$, (iii) $\Omega \in \mathcal{F}$, (iv) $\mathsf{E} \in \mathcal{F}$ implies that the complement of E is in $\mathcal{F}$ (i.e., $\Omega/\mathsf{E} \in \mathcal{F}$), (v) for any sequence of E_1, E_2, ... of elements of $\mathcal{F}$, their union is in $\mathcal{F}$ (i.e., $\mathsf{E}_1 \cup \mathsf{E}_2 \cup \ldots \in \mathcal{F}$). See the following page `http://mathworld.wolfram.com/Sigma-Algebra.html` [Weisstein, 2014].

3. for any $\mathsf{E} \in \mathcal{F}$, the upper bound on $\widehat{\Pr}(\mathsf{E})$ is attainable:

$$\sup_{\Pr \in \mathcal{M}} \Pr(\mathsf{E}) = \overline{\Pr}(\mathsf{E}). \tag{1.30}$$

It follows from the definition that $\overline{\Pr}(\emptyset) = 0$, and $\underline{\Pr}(\Omega) = 1$. Furthermore, for any $\mathsf{E} \in \mathcal{F}$ we have that:

$$\overline{\Pr}(\mathsf{E}) = 1 - \underline{\Pr}(\neg\mathsf{E}), \tag{1.31}$$

where $\neg\mathsf{E}$ is the converse of E. The triplet $(\Omega, \mathcal{F}, \underline{\Pr})$ is called a feasible probability field. ◁

Definition 1.4.2 states the probability theory axioms whereas Definition 1.4.3 builds the concept of a reasonable imprecise probability. That is, the upper bound of the imprecise probability of an event is larger or equal to its lower bound, and the probability of any event is between the bounds. For an imprecise probability to be feasible (Definition 1.4.4), there must exist at least one event with a probability that reaches the lower bound and at least one event with a probability that reaches the upper bound. This last definition means that the bounds of a feasible imprecise probability are as tight as possible. This recalls the concept of bounds consistency we defined for CP (Definition 1.2.19).

Definition 1.4.5 (Uncertain distribution)**.** We call a probability distribution an *uncertain distribution* if at least one of its probability is defined as an interval. ◁

Chapter 2

Coverage and Search Problems: Classical Models and Methodologies

We provide, in this chapter, an overview of the CPP literature and of the OSP literature. We focus on classic algorithms and methods while providing, whenever it is possible, recent applications or ideas in both fields. We introduce CPPs and coverage-related problems in Section 2.1. Then, we present the OSPs in the search theory context in Section 2.2.

2.1 The Coverage Path Planning Problem

CPPs often arise in mobile robotics applications [Choset, 2001]. We may think, for instance, of minesweeping operations [Williams, 2010, Stack and Smith, 2003], seabed surveys in harbors and waterways [Fang and Anstee, 2010], robotic mowing [Weiss-Cohen et al., 2008], harvesting and ploughing [Oksanen and Visala, 2009], marine habitat planning [Galceran and Carreras, 2012], and floor cleaning [de Carvalho et al., 1997] as recent and successful applications of the CPP in mobile robotics. Nonetheless, attempting to cover a physical environment remains an important challenge.

Coverage planning requires assumptions on the robot's capability to sense its environment (sensing capabilities), to know its position (map knowledge and positioning capabilities) and to efficiently plan its path (reasoning and optimization capabilities). Even when assuming perfect positioning and knowledge of the environment in presence of a perfect sensor that allows full coverage of a sub-region in a single scan, a complex path planning problem needs to be solved and discretization choices will influence the overall methodology. Furthermore, mobile robots evolve in different playgrounds (e.g., underwater [Galceran, 2011], in the air [Goerzen et al., 2010], on the ground [Iagnemma and Dubowsky, 2004]) which gives rise to a variety of challenges and plausible applications.

Handling a Continuous Space. Most path planning algorithms use a discretized representation of the continuous environment which is either performed off-line (when the map of the environment is known) or on-line (when the robot discovers the environment by itself). There are two major families of discretization methods [Russell and Norvig, 2013]: roadmaps and cellular decompositions. The general concept of a roadmap is to add points in the (obstacle-)*free space* and to link these points using accessibility links. Two points are linked together if they are neighbors and if the space between them is traversable, i.e., if it is obstacle-free. The concept of neighboring points, which is problem dependant, can intuitively be seen as the nearest neighbors problem in Euclidean space applied to path planning. The *roadmap* (or *skeletonization*) approach to discretization includes *Voronoï diagrams* [de Berg et al., 2008, Aurenhammer, 1991], *visibility graphs* [de Berg et al., 2008], and *probabilistic roadmaps* [Russell and Norvig, 2013]. These approaches are mostly used in an off-line context. More sophisticated methods or adaptation of the usual roadmaps, like the *rapidly-exploring random trees* [LaValle, 1998, LaValle and Jr., 2001], can be performed on-line removing the need of a complete discretization of the continuous environment. Roadmaps, up to a certain point, are graphs where the vertices represent points in the Euclidean space and where the edges represent accessibility. Since the concept of region is more intuitive for coverage (we need to cover regions and not points), cellular decompositions are found more frequently than roadmaps in the coverage literature.

In a *cellular decomposition*, the continuous environment is divided in a set of uniform or non-uniform cells (regions, subregions) [Russell and Norvig, 2013]. *Uniform cellular decompositions* involve grids where cells have the same size. We may think, for instance, of a grid of square or hexagonal cells. If the environment contains obstacles, we have an occupancy grid [Elfes, 1989]. Such a decomposition technique is considered to be *approximate* since some cells may be partially obstructed and/or some parts of the environment may not be fully covered. In a *non-uniform cellular decomposition* the cell size constraint is relaxed. These include trapezoidal decomposition [de Berg et al., 2008], triangulations methods (such as Delaunay triangulation [de Berg et al., 2008]), boustrophedon decomposition (all three exact when considering obstacles), and recursive cellular decomposition [Russell and Norvig, 2013] (approximate). The discretization of a continuous environment using an exact cellular decomposition is $\mathcal{NP}$-hard in the general case of the minimization of the total cut length (i.e., the length of frontier between the cells) [Bast and Hert, 2000]. The same is true for the simpler case of the bisection of a polygon [Koutsoupias et al., 1992]. Huang [2000] presents an alternative and tractable cellular decomposition method that minimizes the number of turns of the robot instead of the cut length.

Optimal exact cellular decomposition (with respect to cell size or to cut length) are computationally hard to achieve. For that reason, simpler cells that are easy to cover (with a lawnmower pattern for instance) are often preferred to decompositions that are carefully performed using some optimality criterion [Choset, 2001]. Moreover, exact decompositions which are non-uniform are often preferred over uniform decompositions in presence of obstacles as they entirely cover the free space [Choset, 2001]. It remains, however, that the larger cells produced by exact decompositions, which are often performed using heuristics, imply assumptions on the robot coverage pattern inside the cell. Such approaches do not result in a low level optimization of the coverage path.

The shape and the orientation of the robot is another issue in the discretization of continuous environments for coverage in mobile robotics. An incorrectly positioned robot may interfere with an obstacle if it is not positioned correctly [Russell and Norvig, 2013]. Handling the robot's shape and orientation forces path planners to work in the configuration space rather than in the original environment. The *configuration space* is the space of the entire robot's possible configurations whose dimensionality depends on the total number of degrees of freedom of the robot. The number of *degrees of freedom* of a robot is the required number of parameters (or directions) needed to describe its complete positioning including the position of its effectors [LaValle, 2006, Russell and Norvig, 2013]. For instance, a square mobile robot that can rotate and translate on a 2 dimensional plane has 3 degrees of freedom whereas it has 2 degrees of freedom if rotations are not allowed. These considerations further add complexity to the discretization problem and simplifying assumptions are often welcomed whenever the study focus on the optimization aspects of a coverage problem rather than on the physical implementation of the method or whenever the application allows for these simplifications.

Finally, even though the concept of a cell is more intuitive than the concept of a point in coverage, roadmaps and cellular decompositions are not completely independent methods and neither of the two methods is ill-suited to coverage. A grid of uniform cells may, for instance, be seen as a roadmap where the points are uniformly distributed over the free space. The same is true for a Voronoï diagram generated from the random points of a probabilistic map.

Map Knowledge. When the environment is known a priori, the problem may be formulated as an *off-line* CPP problem. Otherwise, we have an *on-line* CPP problem where the robot must discover the environment. On-line CPPs, along with suitable real-time (or at least on-line) discretization methods, are often called *sensor-based approaches* whereas off-line CPPs are *map-based.* Sensor-based approaches benefit from fast, simple heuristics rather than on complex optimization algorithms in planning [Choset, 2001]. Map-based CPPs are combinatorial optimization problems.

Positioning. One important assumption usually found in the literature on CPPs is the ability of the robot to precisely know its position. The positioning capabilities of the robot are independent of its map knowledge. A map may be known, but the robot's position in the environment may be uncertain. Similarly, the effectors of the robot may lead to uncertain results [Russell and Norvig, 2013]. This may happens, for instance, in marine robotics where the current influences the robot's trajectory or simply due to imprecise effectors. This is often the practitioner, the researcher or the decision marker's choice to take this uncertainty into account at the path planning level or later during the execution of the coverage plan. These considerations may lead to more complex coverage and navigation models of which [Carlone and Lyons, 2014] is a recent and successful example incorporating mixed-integer linear optimization techniques and models. Uncertain map knowledge and position lead to complex *simultaneous localization and mapping* (SLAM) problems, a capital and actual challenge in robotics [Aulinas et al., 2008].

Sensing. In CPPs with *perfect sensors*, a cell is fully covered after a single scan and no further visits are needed afterward. As such, this *complete* CPP problem is sometimes called an area covering problem [Jimenez et al., 2007] or a region filling problem [Cao et al., 1988]. When a discretization method results in cells that are larger than the range of the sensor, a coarse path is planned to find the global ordering of the traversed regions [Choset, 2001]. A lawnmower pattern is often assumed inside each large cells. Such an approach is required, for instance, when using a boustrophedon decomposition. Depending on the discretization scale and on the range of the sensor, whether limited to its circumference, extended, or infinite, distant cells may or may not be surveyed. Distant cells scanning is allowed, for instance, in [Drabovich, 2008]. A small discretization scale allows for a more precise path planning. At the same time, it leads to a larger combinatorial problem.

In a CPP with *imperfect sensors* it is no longer possible to guarantee a perfect coverage of the environment. The approach is then to add a constraint on the minimal required coverage to be achieved in each cell. The notion of imperfectness of a sensor makes a CPP closer to the search theory field. In some cases, it even involves a similar conditional probability of detecting the search object (i.e., the object for which we are performing the coverage operation) given that it is present in the scanned area [Gage, 1993]. The minimal required coverage constraint along with the conditional probability of detection (or simply imperfect coverage) lead to the need of performing multiple overlapping passes over the environment of interest.

The expression "imperfect sensor" implies applications related to detection and search. It is, however, general. Other interpretations of an imperfect (uncertain) coverage are possible. For instance, an area may require multiple overlapping passes of a cleaning bot if it is known to attract dust. In some cases, multiple visits of a cell are needed due to the optimality criterion or to the constraint of the application. This is the case in the *multirobot-controlled frequency coverage* (MRCFC) problem [Cannata and Sgorbissa, 2011]. A cell may require more than one visit to maintain the distribution of the relative frequency of visits in the environment.

Reasoning and Optimization. There exists a large body of literature on path planning in robotics in four general areas: navigation, coverage, localization and mapping. Algorithms for mobile robotics planning vary in terms of the aforementioned aspects of coverage path planning for which we identified the following modeling questions:

- Is there any readily available map on which to perform the coverage optimization?
- Is a uniform discretization suitable to the application?
- For uniform discretization, what is the discretization scale required by the application?
- Does the robot has any positioning capabilities?
- Is the robot's positioning perfect or can it assumed to be perfect for optimization purposes?
- Is a single visit in each cell sufficient (e.g., for perfect sensors in a surveying operation it is)?
- If multiple visits are necessary, is it due to the sensor's imperfectness or to the application goals?
- How to quantify the imperfectness of a sensor or the notion of imperfect coverage?
- Is it necessary to take imperfectness into account?

Further aspects of reasoning and optimization for coverage problems involve the potential presence of multiple, and possibly heterogeneous agents (including robots), the need for communication, the presence of an evasive target to retrieve by covering or decontaminating the environment of interest, and dynamically changing environments. A thorough review of CPP methods and path planning for navigation are found in the work of Choset [2001] and Paull et al. [2013] respectively.

It remains that mobile robotics applications are one aspect of coverage path planning or coverage problems and that many algorithms for robotic coverage planning are of a combinatorial nature. There are, as presented in the next section, multiple facets to the coverage problem, i.e., to the problem of viewing, surveying, traveling through, or decontaminating an entire space under constraints (may it be a discrete or a continuous space). Many CPP algorithms are taking their inspiration from or use simpler combinatorial optimization problems such as: a TSP (e.g., [Fang and Anstee, 2010], the DpSweeper algorithm of Chapter 3), dynamic programming (e.g., see Chapter 3) or a minimum spanning tree (e.g., [Gabriely and Rimon, 2001]).

2.1.1 The Multiple Facets of the Coverage Problem

The coverage problem (in terms of path planning or not) has many formulations and a long history. It is our intent, in this section, to provide an overview of non-robotics related formalisms and applications of coverage and some variants. The subject of coverage is of interest in many fields related to and/or using (combinatorial) optimization. It is not rare to see algorithms for CPP problems inspired from simpler forms of coverage. We start by reviewing simple (and not necessarily easy) computational geometry and combinatorial coverage problems that are often the components of algorithms that solve coverage problems with a more complex definition, notably in robotics. Then, we do an incursion in the field of pursuit-evasion and related problems. Finally, we review interesting applications of coverage algorithms.

The Bricks to Build Algorithms. Many combinatorial optimization problems have an elegant formulation in terms of graph theory. We described, in Chapter 1, a coverage problem from graph theory: the *Hamiltonian cycle problem* (HCP) which consists in finding a tour that visits each vertex of a graph exactly once. The straightforward extension of the HCP to edged-valued graphs with the additional objective of minimizing the cost of the tour, i.e., the *traveling salesman problem* (TSP), is another example of a graph theory coverage problem. Even though the TSP and the HCP seem to be purely theoretical vehicle and people routing problems, their applications range from gene mapping to program optimization [Cook, 2012]. Industrial applications of the TSP, to name only a few, include circuit boards soldering and drilling, job scheduling and electronic chips testing [Cook, 2012]. The HCP and the TSP received a considerable attention from the scientific community.[1] Many of their extensions and even more specific cases of these problems are proved to be $\mathcal{NP}$-hard including the Euclidean TSP (also called the Geometric TSP) where the distances between vertices are calculated in the plane. The TSP with neighborhood, a generalization of the Euclidean TSP where the vertices are connected regions of the plane, is particularly close to coverage in robotics. A similar TSP generalization, the generalized TSP (which is also called the *covering salesman problem* or the set TSP), partitions the set of vertices of the graph into disjoint subsets of vertices. The goal of a generalized TSP is to find a tour of minimal cost that visits all subsets at least once. It is also worth mentioning that the HCP on grid-graph is $\mathcal{NP}$-hard [Itai et al., 1982]. Intuitively, a grid-graph is a grid of uniform square cells with holes. In a square cell decomposition in mobile robotics, the holes might, for instance, represent obstacles. Many circuit planning problems have a path planning counterpart. The latter is often as hard as the former (e.g., the *Hamiltonian path problem*).

Another well-know coverage problem on valued graphs is the *minimum spanning tree* (MST) problem [Graham and Hell, 1985]. We mentioned the MST problem in Section 2.1 on CPP

[1] Both the HCP and the TSP are fundamental combinatorial optimization problems. While the HCP is attributed to W. R. Hamilton, the origin of the TSP is unclear. An historical treatment of the matter is provided by [Schrijver, 2005].

for mobile robotics [Gabriely and Rimon, 2001]. Given a valued graph G, the MST problem consists in finding a connected subgraph T of G that has no cycle (i.e., T is a subtree of G), that shares the same vertex set as G and that has a minimal cost. The MST problem is tractable making it possible [Graham and Hell, 1985] for a robot to ensure the complete coverage (with perfect sensor and limited detection range) of a grid of uniform cells in polynomial-time [Gabriely and Rimon, 2001].

Similar coverage problems arise in computational geometry. The *art gallery problem* which consists in positioning stationary watchmen (or cameras) inside a continuous environment defined by a polygon with visibility-obstructing holes is one of them. There are strong links between the art gallery problem and the discretization by visibility graph mentioned in Section 2.1. Whenever the guards are restricted to be positioned on the vertices of the polygon, the problem becomes a minimal dominating set problem which is a combinatorial optimization problem from graph theory. Given a graph, the *minimal dominating set problem* consists in finding a vertex set $\mathcal{D} \subseteq \mathcal{V}(G)$ of minimal cardinality such that all vertices not in $\mathcal{D}$ are guarded, i.e., for all $x \in \mathcal{V}(G) \setminus \mathcal{D}$ there exists a vertex $x \in \mathcal{D}$ such that $\{d, x\}$ is an edge of the graph. Whereas the minimal dominating set problem consists in minimizing the cardinality of set $\mathcal{D}$, the decision version, i.e., the dominating set problem, consists in determining whether or not there exists a dominating set $\mathcal{D}$ such that $|\mathcal{D}| \leq k$ where k is a parameter of the problem. Two interesting counterparts of the dominating set problem are the *connected dominating set problem* [Mahalingam, 2005, Karami et al., 2012-01], which requires any vertex in $\mathcal{D}$ to share an edge with at least another vertex in $\mathcal{D}$, and the *dominating path problem* [Faudree et al., 2014], which precisely requires that $\mathcal{D}$ forms a path on the graph. The art gallery problem with a mobile watchman is called the *watchman route problem* [Chin and Ntafos, 1988]. The usual objective is to find the shortest path that enables the watchman to cover the whole environment. Both the art gallery problem and the optimum watchman route problem are $\mathcal{NP}$-hard in the general case [de Berg et al., 2008, Chin and Ntafos, 1988]. Variants involve a limited visibility range for the watchman and various constraints on the space to survey [Ntafos, 1992]. Some specific cases, e.g., the case of simple polygons, are tractable [Chin and Ntafos, 1988, Ntafos, 1992, Carlsson et al., 1999].

Handling Evading and Possibly Smart Targets. The coverage problem (and the CPP) formulations considered up to this point, if taken in a target detection or a surveillance context, were based on the assumption of a stationary target. We now provide an overview of formalisms that deal with one or many evasive targets. From a search theory point of view (the point of view taken in Section 2.2), problems that deal with smart objects or targets are called *search games*. These include both evasive and cooperative targets. The point of view we adopt here concerns non-cooperative targets, i.e., pursuit-evasion problems (or games).

Pursuit-evasion problems are easily viewed from a graph theoretic point of view. In a first variation, the pursuit-evasion problem is formulated as a game with two players: a *cop* and a *robber*. Players take turns starting with the cop who chooses a vertex on the graph. This is her starting position.[2] The robber plays next choosing his starting vertex. The goal of the game is for the cop to catch the robber in a finite amount of time. Then, at her/his respective turn, the player moves along an edge from her/his current vertex to a neighbor. The original game is played in total information. First introduced by Nowakowski and Winkler [1983] and Quilliot [1983], the cop and robber game is not only a game but a way to characterize graphs. On some graphs, there exists a strategy for the cop to win the game by catching the robber in finite time. On others, the robber always wins if he plays perfectly. We might also want to consider a fast robber who moves, in a single turn, to any accessible vertex from his position [Marcoux, 2014]. A vertex is *accessible* to the robber if there exists an unguarded path between his current vertex and that vertex. We say that a path is *guarded* if it crosses any vertex in the cop's neighborhood. Other variants, relaxing the worst-case assumption on the robber's motion, consider an inebriated robber. The robber is now drunk [Kehagias and Prałat, 2012, Komarov and Winkler, 2013, Simard et al., 2014, 2015] and/or invisible [Kehagias et al., 2013] and he has a motion model which may be Markovian or totally random. The problem of determining whether or not a cop has a winning strategy on a given graph and its k-cops variant are tractable when the game is played in total information [Clarke and MacGillivray, 2012, Hahn and MacGillivray, 2006].

The case of an invisible evader is also taken into account in *graph searching* problems. The objective, in graph searching, is to minimize the number of agents (or searchers) needed to decontaminate (or clear) a given graph [Parsons, 1978]. To *decontaminate* an edge $\{u, v\}$, a searcher standing on vertex u *slides* to vertex v. A *recontamination* of an edge $\{u, v\}$ occurs if, at any moment, it can be connected by a searcher-free path to a contaminated edge. The graph is *cleared* if all edges are decontaminated. Again, variants are numerous. Parsons [1978] formulation is called *edge-search*. Edge-search is $\mathcal{NP}$-hard [Megiddo et al., 1988]. A recent annotated bibliography of graph searching problems (including the cop and robber games) is found in the work of Foming and Thilikos [2008].

A framework for coverage in mobile robotics in presence of an evasive target and using similar "graph-searching" concepts is *graph-clear* [Kolling and Carpin, 2007, 2010]. In a graph-clear problem, the edges and the vertices of the graph can be either cleared or contaminated. Cleared vertices and edges can be recontaminated if, at any time, it can be connected by a searcher-free path to a contaminated edge or vertex. Given a contaminated graph, the objective is to find a strategy of minimal cost to clear the graph. A *strategy* associates a number of searchers to each edge and vertex at each time step until, at some point, the graph is cleared. The *cost* of a strategy is the maximum number of searchers needed at any time step

[2] Without loss of generality and to avoid confusion, the cop is historically played by a woman whereas the robber is played by a man.

on the graph. Limited range [Kolling and Carpin, 2009b] and probabilistic detection [Kolling and Carpin, 2009a] are considered in later versions of the framework. Graph-clear is $\mathcal{NP}$-hard [Kolling and Carpin, 2007].

Coverage and Geographic Information Systems. A common use case of a *geographic information systems* (GIS) is to perform a visibility-based terrain analysis [De Floriani and Magillo, 2003]. In surveillance for instance, it is important to know the area visible from a set of points on a map. This visible area, taking into account terrain, obstacles, and possibly weather conditions, is a *viewshed*. GIS viewshed tools are used in security monitoring (surveillance) to know the area visible from sensors or cameras locations [Murray et al., 2007]. They are also used to position other features such as buildings or radio masts [Kim et al., 2004]. Trying to find the optimal position of a set of sensors or features with respect to some visibility criterion (e.g., maximizing the visibility of a fixed number of sensors, or minimizing the number of sensors for total coverage of an area [Goodchild and Lee, 1989]) or to multiple criteria often give birth to large instances of hard combinatorial optimization problems. All of these are different forms of the coverage problem.

2.2 The Optimal Search Path Problem

Search problems arise in many applications related to detection. Recent examples of applications of search problems are found in search and rescue [Abi-Zeid et al., 2011b, Breivik et al., 2012, Berger et al., 2012, 2013, Guitouni and Masri, 2014, Berger and Lo, 2015], military surveillance [Lim and Bang, 2010], discovery of services and facilities in presence of potential disruptions [Berman et al., 2011], malicious code detection [Kranakis et al., 2007], covert messages (violating the security policies of the system) on the Internet [Chandramouli, 2004], locating a mobile user in a cellular network for optimal paging [Verkama, 1996], and even in mobile robotics [Chung et al., 2011, Joho et al., 2011, Macwan, 2013, Kulich et al., 2014] we discussed in terms of coverage problems in the previous section.

The OSP problem is a search path planning problem that emerged from search theory, an operations research discipline. A short review of the key concepts from search theory is useful to understand the OSP problem. Stone [2004] provides a complete introduction to the field. An intuitive introduction to search theory is provided by Frost [1999a,b,c,d]. We provide the necessary background to fully understand the OSP in the next section.

2.2.1 From Search Theory to Search Path Planning

During the Battle of the Atlantic (1939-1945), the Anti-Submarine Warfare Operations Research Group (ASWORG) was mandated to enhance U-boats detection methodologies. They defined the notion of search effort allocation. That is, the amount of deployed resources in each part of an area needed to locate a search *object* (target). The resources spent in the action of searching may be of different types depending on the application (goals and constraints of the decision maker) and the formulation (objective functions and hypotheses). Broadly speaking, *search effort* is of one of three types [Stewart, 1979]:

1. an *infinitely divisible* search effort is continuous and may be allocated to multiple regions subject to the problem's specific constraints (e.g., time, amount of fuel);
2. an *arbitrarily divisible* search effort is discrete and may be allocated to multiple regions subject to the problem's specific allocation constraints (e.g., available sensor's scans, available aircrafts);
3. an *indivisible* search effort amount must be allocated to a single region (e.g., one aircraft, one unique searcher).

The area of interest, we call the search *environment*, is a set of continuous sub-*regions*. By allocating more or less effort to specific regions with respect to a searcher's (or multiple searchers) allocation constraints and resources constraints, we define a search plan. The *search plan* may be either a sequence of regions, whenever the specific time of a search is important to know (e.g., for mobile objects), or a set of regions, whenever the time does not affect the outcome (e.g., for immobile or very slow objects). A search plan may, for instance, be constrained to be a path such as in OSPs or it may be made of a set of non-overlapping rectangular areas assigned to airborne search units for sweeping such as in the *multiple rectangular search areas problem* [Discenza, 1979, Abi-Zeid et al., 2011b]. A common criterion for an *optimal* search plan is to maximize the *cumulative overall probability of success* (*cos*). This optimality criterion depends on the prior knowledge on the location of the object we search for, possibly on its motion model if it is moving, and on its *detectability* since most sensors used for searching are *imperfect*.

The object's exact location in the search environment is a priori unknown. However, its whereabouts are known. The whereabouts of the object are represented by a probability distribution over the regions of the environment, i.e., the *probability of containment* (*poc*) distribution. Searched regions often have a large area in search theory. Searching at sea involves, for example, environments discretized by grids where the uniform cells representing the regions have an area of several square nautical miles [Abi-Zeid and Frost, 2005]. For that reason, the searcher is often not able to survey the entire region with a single scan and it is assumed that the visit of each region involves sweeping that region with respect to an assumed search pattern. The same is true for smaller regions: a thorough search intuitively involves a sweep of the region.[3] We see, in Figure 2.1, a search plan example over a search environment discretized by a grid. The search plan we have here is a path of length 6. We

[3] We can think of how we visually search (sweep) for a missing object in a room. This search action often involves a sweep of the area even though the pattern we employ might be a random one.

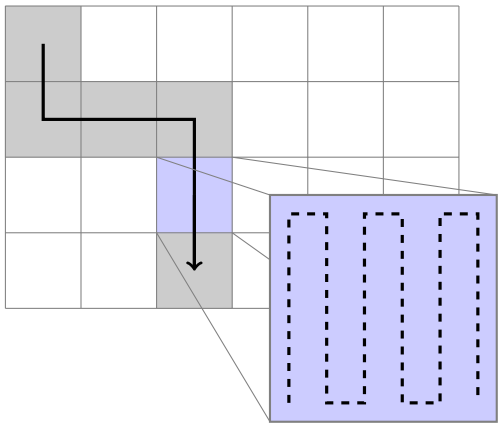
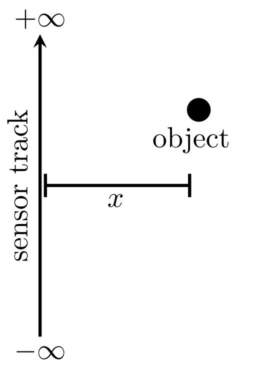
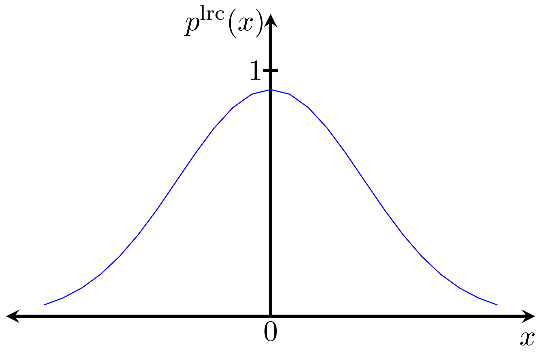


Figure 2.1: A search environment discretized by a grid along with a path-like search plan of length 6; a given search pattern is assumed in each region. Filled regions are swept by the sensor. The search pattern in the blue filled region is represented by a dashed line.



(a) A single sensor track



(b) A lateral range curve (fictional)

Figure 2.2: An illustration of the concept of lateral range curve

might suppose, for instance, that 6 units of effort were available for searching and that each of these effort units were allocated to different regions under path constraints. We suppose, in this example, that the search pattern is a zigzag with uniformly spaced tracks.

In practice, the search pattern is important as it influences the probability of finding what we are searching for. We need, to understand the implications of the search pattern on this probability, to define the concept of detectability index (also called *sweep width*). Even with a broad range sensor, the distance from the object to the sensor influences the lateral detection probability. Each sensor is characterized by a lateral range curve. Suppose that a given sensor travels on a straight path at a constant speed as in Figure 2.2(a). Then, we may compute, using several passes on this track, its probability of detecting an object located at a distance x from that track. If we do the same for all x, we obtain a curve: the right hand-side lateral range curve of the sensor. The *lateral range curve* $p^{\text{lrc}}(x)$ is the instantaneous probability of detecting a given object under specific conditions as a function of the lateral distance

x from the sensor to the object ($-x$ for an object located to its left) (see Figure 2.2(b)). From our knowledge on the lateral range curve, we compute the object's detectability index. The *detectability index* is defined as the integral of $p^{\mathrm{lrc}}(x)$ over x on the interval $[-\infty, \infty]$, $\int_{-\infty}^{\infty} p^{\mathrm{lrc}}(x)\,\mathrm{d}x$.

Just as the sweep width of the sensor influences detection so does the assumed search pattern of the sensor over the continuous region. This assumption is a component of the searcher's detection model. It is also a key aspect used in search theory to determine what our *pod* function will look like. From that search pattern follows the concept of *detection law*. A widely used detection law is the exponential one [Stone, 1983]. An exponential detection law bounds below the detection probability attainable by any other search pattern in a given region. It is a worst-case assumption corresponding to a random search pattern. An example of exponential detection law taking into account the amount of effort e given a sweep width $\omega(r)$ with r being a region of the environment is the following:

$$pod(e, r) = 1 - \exp(-\omega(r) \cdot e). \tag{2.1}$$

As $\omega(r)$ grows, the detection probability increases. The same is true for the amount of effort e. It can also be seen that the probability of detection function we defined as an example follows the law of the diminishing returns.

In practice, the detection law is used to compute the *pod* function, also called detection model, we summarize as a function of:

- the effort spent over a region in the action of searching over a defined period;
- the searched region (to take into account factors such as terrain topology and vegetation); and possibly,
- the time of the search (to take into account factors such as weather and exhaustion).

The last two factors are taken into account when computing the object's detectability index ω which is an important parameter in the detection models of a searcher for practical search operations. In practice, sweep width tables are computed under very specific terrain and weather conditions to allow for an efficient search (e.g., [Koester et al., 2004]). In empirical experiments from the literature focussing on problem solving rather than on modeling, the sweep width is often fixed over the entire environment. Furthermore, it is not uncommon for researchers to reduce the detection model to a single probability value (e.g., [Washburn, 1983, Lau et al., 2008, Sato and Royset, 2010]) whenever the emphasis of the study is on the performance of a problem solving technique rather than on a novel detection model. This practice leads to a simplified detection model in empirical experiments.

Search theory does not limit itself to immobile objects. Moving objects are characterized by a *motion model*. These models are classified in two families: models for passive objects and models for active objects. Active objects may try to meet or to evade the searchers. In this case, the search problem is often called a search game or a *two-sided* search. Passive objects just moves around under the assumptions of the model. In this case, the search problem is *one-sided* in that the object does not respond directly to the searcher's actions. Detection search problems with a stationary object, which is considered passive, belong to the family of one-sided search problems.

Search games motion models are often worst-case (e.g., pursuit-evasion problems) or best-case (e.g., *rendez-vous* search) scenarios where the object is assumed to have full knowledge of the searcher's actions. The literature on search games is vast with in-between cases involving partial knowledge and inter-visibility. We will touch on key search games concepts only where it is helpful to the understanding of our search problems. The OSP problem is historically a one-sided search problem. The motion model is a known parameter of the problem. Our primary focus is OSP problem variants where the *cos* is the optimality criterion.[4] We limit this review to the class of OSP problems formulated in discrete time and space with a single object. We do not consider decentralized approaches, trajectory planning, false detections or false targets and we restrain ourselves to the search theory literature as opposed to path planning in the robotics literature we reviewed in Section 2.1.

2.2.2 Searching for a Stationary Object

In this section, we introduce the formalism of the OSP problem with a stationary object. We assume that the continuous search environment is discretized by a graph. Each vertex of the graph represents a continuous (sub-)region of the original environment. The edges of the graph represent the searcher's accessibility constraints. Graphs are usually understood to be undirected and irreflexive, i.e., without loops. For the purpose of modeling the search environment, we assume reflexive graphs. Reflexive graphs are natural in the context of detection search problems. The loops on the vertices of a reflexive graph enable the searcher and the object to stay at their current location instead of moving on. We assume undirected reflexive graphs although the approach is general.

Let $G_A = (\mathcal{V}(G_A), \mathcal{E}(G_A))$ be the accessibility graph that represents the search environment where $\mathcal{V}(G_A)$ is a set of discrete regions. A vertex r is accessible from vertex s if and only if the edge (s, r) belongs to the accessibility graph G_A. Let $y_t \in \mathcal{V}(G_A)$ be the searcher's location at time $t \in \{1, \ldots, T\}$. When $y_t = r$, we say that vertex r is searched at time t with an associated probability of detection. A search plan P (i.e., the sequence of searched

[4]Alternate terminology uses the following terms: "probability of detection" instead of "probability of success", "glimpse probability" instead *pod* and/or "whereabouts" instead of "probability of containment".

vertices) is determined by the searcher's path on G_A starting at location $y_0 \in \mathcal{V}(G_A)$:

$$P = [y_0, y_1, \ldots, y_T], \tag{2.2}$$

where T is the maximal search plan length (in number of time steps). The usual search theory definition of optimality is to obtain the search plan of maximal *cos* that respects the physical and operational constraints of the searcher. An OSP makes no exception on the objective.

The unknown object's location is characterized by a probability of containment (*poc*) distribution over $\mathcal{V}(G_A)$ that evolves in time due to the searcher's knowledge updates following unsuccessful searches. A local probability of success (*pos*) is associated with the searcher being located in vertex r at time t. It is the probability of detecting the object in vertex r at time t defined as:

$$pos_t(r) = poc_t(r) \times pod(r), \tag{2.3}$$

where $pod(r)$ is conditional to the presence of the object in r and is assumed independent of past searches. This detection model is known a priori. For all $t \in \{1, \ldots, T\}$, $r \in \mathcal{V}(G_A)$, the detection model is

$$pod(r) \in (0, 1], \qquad \text{if } y_t = r; \tag{2.4}$$

$$pod(r) = 0, \qquad \text{otherwise.} \tag{2.5}$$

The OSP formalism assumes that a positive detection of the object stops the search. The probabilities of containment change in time according to the negative information collected on the object's presence by the searcher. Thus, for all time $t \in \{2, \ldots, T\}$, we have that

$$poc_t(r) = poc_{t-1}(r) - pos_{t-1}(r) = poc_{t-1}(r)\,(1 - pod(r)). \tag{2.6}$$

As mentioned, the optimality criterion for a search plan P is the maximization of the global and cumulative success probability of the operation (*cos*) over all vertices and time steps defined as:

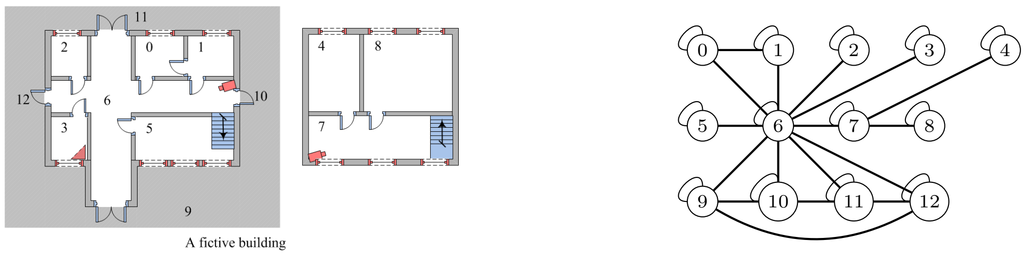
$$cos(P) = \sum_{t \in \{1,\ldots,T\}} \sum_{r \in \mathcal{V}(G_A)} pos_t(r). \tag{2.7}$$

The objective value at time step t, i.e., cos_t, is

$$cos_t = \sum_{t' \leq t} \sum_{r \in \mathcal{V}(G_A)} pos_{t'}(r). \tag{2.8}$$

An optimal search plan P^* maximizes $cos_T(P^*)$ we often note as $cos(P^*)$.

Example 2.2.1 (Searching for a stationary object)**.** Figure 2.3(a) shows the example of an environment with doors and stairs accessibility. The accessibility graph as illustrated on



(a) A building map (b) An accessibility graph

Figure 2.3: A structured search environment with a stationary object

Table 2.1: The probability of containment of each vertex at each time step and the cumulative overall probability of success for each time step for the search plan P^* of the example of Figure 2.3.

t	Probability of containment in vertex r at time t ($poc_t(r)$)													$cos_t(P^*)$
	0	1	2	3	4	5	6	7	8	9	10	11	12	
1	-	-	-	-	.3	.5	-	.1	.1	-	-	-	-	0
2	-	-	-	-	.3	.05	-	.1	.1	-	-	-	-	.45
3	-	-	-	-	.3	.05	-	.1	.1	-	-	-	-	.45
4	-	-	-	-	.3	.05	-	.01	.1	-	-	-	-	.54
5	-	-	-	-	.03	.05	-	.01	.1	-	-	-	-	.81

Figure 2.3(b) follows directly. Regions are numbered from 0 to 12. Suppose that we have the following initial containment probabilities:

$$\begin{aligned} poc_1 &= [poc_1(0), poc_1(1), \ldots, poc_1(12)] \\ &= [0, 0, 0, 0, .3, .5, 0, .1, .1, 0, 0, 0, 0]\,. \end{aligned} \tag{2.9}$$

The searcher, starting in vertex $y_0 = 3$, needs to maximize her/his probability of finding the object in $T = 5$ time steps. We assume that $pod(y_t) = 0.9$ for all $t \in \{1, \ldots, T\}$. The object is stationary. An optimal search plan P^* for that task would be:

$$P^* = [y_0, y_1, \ldots, y_5] = [3, 6, 5, 6, 7, 4]\,. \tag{2.10}$$

Using Table 2.1, we explain why search plan P^* is optimal. Starting from vertex 3, the searcher first moves to vertex 6. This move enables the searcher to reach vertices 4, 5, 7 and/or 8 where the probability of containment is non-null. With 5 time steps available for searching, it is impossible to visit all vertices with non-null probability. Vertices 4 and 5 are the ones with the highest containment probability. The searcher goes in 5. Her/his gain in vertex 5 is a local probability of success of .45. Whenever the searcher visits a vertex, the probability of containment is updated. After the update, vertex 5 is no longer interesting since its probability of containment becomes .05. The shortest path to vertex 4 is through vertices 6 and 7. There is no gain in visiting vertex 6. There is a gain of 0.09 in vertex 7.

Following the visit of vertex 7, its probability of containment is updated to .01. Finally, the searcher reaches vertex 4 where her/his local probability of success is .27. The objective (*cos*) value of the optimal search plan P^* is equal to 0.81. ◁

Even though the $\mathcal{NP}$-hardness of the OSP problem with a stationary object is a well-known result [Trummel and Weisinger, 1986], we should point out that the main difficulty is with the searcher's accessibility constraints under a limited search time constraint. If the cost of moving between vertices is null, then the problem is solved by a greedy allocation of the search effort over the environment. The same will be true whenever the accessibility graph is a complete graph. This interesting observation is not completely novel. Although stated differently, such a greedy effort allocation, called an incremental search plan [Stone, 1983], has been proved optimal.

2.2.3 Searching for a Moving Object

Markovian motion models are widely used in the OSP literature. We restrict our overview to this type of object's motion models. When the object is moving according to a known Markovian motion model, the only modification to the OSP formalism with a stationary object is in the probabilities of containment update equation (2.6). Let $\mathbf{M}_{sr}$ be the probability of the object moving from vertex s to vertex r within one time step. With a moving object, the probabilities of containment update equation depends on both the motion model of the object and the negative information collected while searching. We define this update equation as:

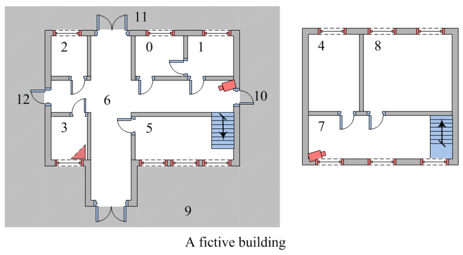
$$poc_t(r) = \sum_{s \in \mathcal{V}(G_A)} \mathbf{M}_{sr} \left[poc_{t-1}(s) - pos_{t-1}(s) \right] . \tag{2.11}$$

Example 2.2.2 (Searching for a moving object)**.** Figure 2.4(a) shows the example of an environment with doors and stairs accessibility. Suppose that $T = 5$, $y_0 = 3$, $poc_1(4) = 1.0$, and $pod(y_t) = 0.9$ for all $t \in \{1, \ldots, T\}$. Assuming that the object follows a Markovian motion model uniform between accessible vertices 2.4(b), an optimal search plan P^* would be

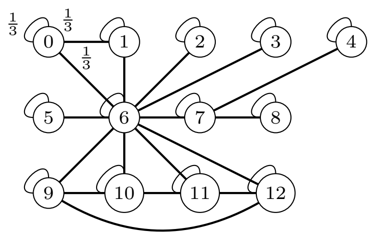
$$P^* = [y_0, y_1, \ldots, y_5] = [3, 6, 7, 7, 7, 7] . \tag{2.12}$$

Using Table 2.2, we explain why search plan P^* is optimal. Starting from vertex 3, the searcher first moves to vertex 6 since the probability of containment is high in vertex 4. Then, the only accessible vertex where the probability of containment is nonzero is vertex 7. Therefore, the searcher moves from vertex 6 to vertex 7. Finally, the search plan stabilizes in vertex 7 since it has the highest probability of containment at each subsequent time step. The objective value is computed as follows:

- compute the local probability of success in vertex y_1 at time step 1 ($pos_t(y_1)$) using Equation (2.3);



(a) A building map



(b) An accessibility graph with a uniform Markovian motion model (displayed for vertex 0 only)

Figure 2.4: A structured search environment with a moving object

Table 2.2: The probability of containment of each vertex at each time step and the cumulative overall probability of success for each time step for the search plan P^* of the example of Figure 2.4. The probabilities are rounded to the third decimal.

t	Probability of containment in vertex r at time t ($poc_t(r)$)													$cos_t(P^*)$
	0	1	2	3	4	5	6	7	8	9	10	11	12	
1	-	-	-	-	1	-	-	-	-	-	-	-	-	0
2	-	-	-	-	.500	-	-	.050	-	-	-	-	-	.450
3	-	-	-	-	.263	-	.012	.026	.012	-	-	-	-	.686
4	.001	.001	.001	.001	.138	.001	.008	.015	.013	.001	.001	.001	.001	.817
5	.001	.001	.001	.001	.073	.001	.008	.008	.01	.002	.002	.002	.002	.889

- for all vertices r, compute the probability of containment at time step 2 ($poc_2(r)$) using Equation (2.11);
- apply the same process for time steps 2 to T;
- sum all the local success probabilities obtained in time steps 1 to T to compute the objective (cos) value of the search plan P.

The objective (cos) value of the optimal search plan P^* is equal to 0.889. $\triangleleft$

As a generalization of the OSP problem with a stationary object, the OSP problem with a moving object and a single searcher is also $\mathcal{NP}$-hard [Eagle and Yee, 1990]. Prior to 1998, most work on the single searcher OSP problem in discrete time and space involved B&B algorithms [Washburn, 1998]. Stewart [1979] is often considered to be the first direct work on the OSP problem in discrete time and space from a search theory perspective.[5] In a first formulation of the problem, Stewart [1979] considered a moving object and an infinitely divisible search effort. The problem was solved to optimality using a network flow under the

[5]A review of previous formulations where the searcher and the object are restricted to one dimension can be found in [Benkoski et al., 1991].

assumption of an exponential detection law. This first formulation is not to be confused with a single searcher OSP problem since the effort can be divided over many search paths.

In the indivisible effort case (i.e., the single searcher case), a depth-first B&B algorithm was proposed [Stewart, 1979]. The bounding procedure involved a relaxed problem in which the regions' adjacency constraints are replaced by accessibility constraints. The relaxed problem was solved using Brown's algorithm [Brown, 1980]. As a result, the bound does not guarantee the optimality of the B&B algorithm: Brown's algorithm guarantees optimality only in the infinitely divisible effort case. Eagle [1984] considered a Markovian object's motion model and proposed a dynamic programming approach. Eagle and Yee [1990] presented an optimal bound for the B&B algorithm. With an object following a Markovian motion model and an exponential *probability of detection* (*pod*) function, their approach produced an optimal bound by relaxing the search effort indivisibility constraint over a set of vertices while maintaining the path constraints. The bound is computed in polynomial time. Martins [1993] proposed the MEAN bound which is based on the maximization of the expected number of successes in the remaining time. The bound is computed using the longest path problem on a directed acyclic graph where the cost of each arc is a function of the remaining *poc* value at a specific time. Lau et al. [2008] proposed the DMEAN bound which was derived from the MEAN bound [Martins, 1993]. The DMEAN bound value is obtained by solving a longest path problem similar to the one of Martins [1993] where the cost of each arc is discounted by the last obtained *pos* value. Simard et al. [2014, 2015] proposed a bound based on a search game relaxation of the OSP into a Markov decision process [Russell and Norvig, 2013] which is inspired from the total detection (TD) heuristic we present in Chapter 4.

In the arbitrarily divisible search effort case, Stewart [1979] proposed a sequential search effort allocation for small effort amounts and a relaxation of the effort indivisibility constraints for a sufficiently large effort. Both the arbitrarily divisible and the infinitely divisible search effort formulations allowed the effort to be divided among several paths. An arbitrarily divisible effort of 3 may, for instance, allows 3 different searcher's path. This is a form of multiple searchers OSP problem. While the arbitrarily divisible effort case [Stewart, 1979] is an implicit form of multiple searchers OSP model, Dell et al. [1996] extended explicitly the OSP problem with a moving object to the multiple searchers case. Furthermore, they applied the MEAN bound [Martins, 1993] to the case of multiple searchers. Since the complexity of the multiple searchers OSP problem grows exponentially in the number of searchers, they developed, in the same paper, six heuristics for the multiple searchers OSP problems. The first method used as a benchmark involved a local search procedure with random restarts. Two heuristics were based on the total number of expected successes. Two were genetic algorithms. The last one was a moving horizon heuristic that solved a sequence of simplified problems with $T' < T$ time steps.

Among the recent developments linked to the generalization of the OSP problem with a single searcher, Lau [2007] and Lau et al. [2008] introduced the OSP problem with a non-uniform travel time between connected regions (OSPT). In this formulation, the searcher does not search while traversing an edge to another region. Lau et al. [2006] and Lau [2007] further generalized the OSP model to allow detections during travel time. The DMEAN bound, developed for the OSP, is applicable to the OSPT and to the generalized OSP. Lau [2007] also investigated a special case of OSP problem with multiple searchers aided by scouts where a successful detection of the object by a scout does not end the search but adds positive information for searchers. Sato [2008] and Sato and Royset [2010] generalized the OSP problem (they renamed the *single searcher path* (SSP) problem) to the case of path dependent detection model and introduced additional resource constraints. The new problem is called resource-constrained SSP (RSSP). New bounding techniques and heuristics based on network reduction, on the longest path and on Lagrangian relaxation are presented for both the SSP and the RSSP problem. They worked on the B&B algorithm to further adapt it to the multiple searchers case as well. The OSP problem with visibility (OSPV) was also proposed to allow distant effort allocation from a single search point in the OSP problem formalism [Morin, 2010, Morin et al., 2009, 2010]. The cases of indivisible and arbitrarily divisible search effort are considered from a single attribution source, i.e., a single search path. The problem was tackled using mixed-integer linear programming [Morin et al., 2009] and ant colony optimization [Morin, 2010, Morin et al., 2010].

2.3 Coverage and Detection Search Problems

We adopt, in this thesis, the point of view of search operations for both coverage and detection search problems. This leads to an unified view of both families of problems that could help a decision maker in choosing which formalism better suits the practical context of the decision to be made. For instance, which formalism between a CPP and an OSP variant is better suited to detect a mine? The answer is contextual and depends on both the constraints and the goal of the decision maker.

A decision maker that has insights on the whereabouts of the mine may represent it as a probability distribution. In this context, search theory known formalisms, models, and algorithms already proved to be useful in practice. We can think, for example, of the use (and birth) of search theory during World War II [Koopman, 1956a,b, 1957]. Another classic case is that of the SS Central America. The vessel sank in 1857 with approximately 400 million dollars of gold bars and coins on board. Years later, in 1985, search theory was applied to the case. The wreck, lost for more than a century, was found [Stone, 1992]. Nowadays, classic and novel models and algorithms from the theory of optimal search evolve into complete systems of which SAROPS [Netsch, 2004] and SARPlan [Abi-Zeid and Frost, 2005] are two successful examples. On the other hand, cases may occur where the whereabouts are totally unknown. Replacing the knowledge on the whereabouts by a uniform distribution is a strong assumption. Using a uniform distribution is assuming that the mine may be anywhere with uniform probability. Even if the approach is, under some circumstances, mathematically correct, it does not represent the reality: the whereabouts are unknown, the probabilities are unknown. In this context, it is better to look everywhere as fast as we can, i.e., to use a coverage path planning formalism that minimizes the length of the path [Drabovich, 2008].

It can be seen, from our overview of the literature on both coverage and (detection) search, that our two chosen path planning problems, namely the CPP and the OSP, are relatively close to each other. There are, however, some welcomed differences between the two problems. These differences are welcomed since both problems have a tendency to complete each other rather than being clearly incompatible. As discussed in the minesweeping example, one particular point that justifies these differences is the absence of priors on the location of the search object in most CPPs. This is easily justified by the absence of any object to search for in some applications of coverage path planning (e.g., for cleaning). This can be, from a search, surveillance, or detection perspective surprising. However, it remains that this particular point makes a CPP a method of choice when the decision maker does not have any knowledge on the plausible location of the object which is a frequent case. This has also the additional benefit of simplifying the formulation of the problem.

Without any prior on the location of the object it also makes sense to aim at fully covering the environment while minimizing the expenses in which case the term “expenses” may mean, for instance, the time spent in the action of searching the environment which time is easily expressed in terms of path length or cost. Working under a time constraint as in an OSP problem, instead of under a time objective, makes less sense when almost any search action will result in the same output: a cell is covered. It is also interesting to notice how, in CPPs, the notion of coverage acts as a constraint whereas it is not the case in an OSP problem. In OSPs, the searcher aims at maximizing some form of coverage called the *cos*. This is done under a time (expense) constraint rather than under a time (expense) criterion. Finally, coverage problems sometimes include a notion of sensor imperfectness that makes them particularly close to OSP problems. This is the case, for example, of the CPPIED problem we fully describe in the next chapter.

Chapter 3

Coverage Path Planning to Search without Prior

This chapter is based on our original work published in [Morin et al., 2013b] and presented at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013).

We focus, in this chapter, on the *coverage path planning* (CPP) problem in the context of deploying mobile *autonomous underwater vehicles* (AUVs) for mine countermeasures. Our specific application consists of planning the path of an AUV from the REMUS family of vehicles [Nicholson and Healey, 2008], designed to swim long distances at constant speed and altitude with infrequent turns, and with no or very little capacity to sense and avoid obstacles [Fang and Anstee, 2010]. Our AUV is fitted with a navigation system to adjust its position. It is also equipped with sidescan sonar to search for mines on the bottom of the ocean [Reed et al., 2003]. Figure 3.1 presents a sidescan sonar towed by a ship. There is a gap produced by the sonar directly under the tow vehicle. No detection occurs in this blind spot. Without loss of generality, we assume that there are no blind spots, since the use of "gap filler" forward looking sonars has become prevalent.

Sidescan sonars, like many sensors, are imperfect. We consider, in our formalism, no false positives. Similarly to the search for a lost vessel [Stone, 1992], any positive detection must be investigated in our minesweeping context. Just as in search problems from search theory, we assume that the conditional probability of detecting a target of interest given that it is within the sensor's range is less than 100% [Gage, 1993]. This conditional probability of detection can be interpreted as the degree of coverage resulting from sensor or actuator imperfectness [Gage, 1995]. While traveling on a path segment, the AUV surveys a fixed distance sideways with a varying conditional probability of detection that depends on the seabed type of the surveyed region (e.g., complex seabed, sand ripples, flat seabed), and on the range of the sensor. A given minimal coverage (minimal conditional probability of detection) must be achieved over the whole area of interest for the path to be *feasible*. Since the seabed map is available as a

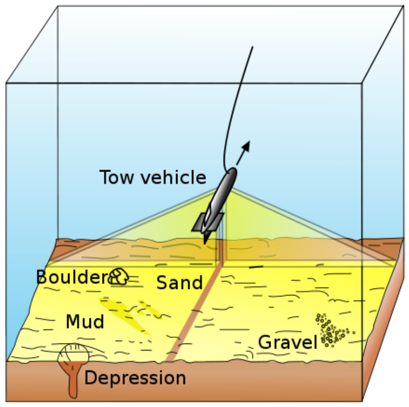


Figure 3.1: A sidescan sonar representation; image taken from [Wikipedia, 2013]

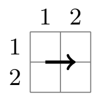
grid of uniform square cells, we have an off-line CPP problem where the discretization scale is such that the side of a cell is not larger than the sonar's range. Time to completion is an important issue. Longer paths to cover an area take more time and are more expensive to complete. Turning also takes more time and may increase navigational errors [Choset, 2001]. Therefore, the goal is to find a feasible path that minimizes the total traveled distance and the total number of turns as two separate criteria and in that order.

The problem presented in this chapter differs from classical CPP problems in three ways:

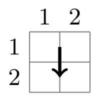
- first, the sensors are imperfect;
- second, the range of detection is not limited to the location of the AUV and varies with the distance separating it from the scanned area, and with the seabed type;
- third, we do not require perfect coverage, we rather must ensure a minimum coverage level everywhere in the area of interest.

We call this formulation, the CPP *with imperfect extended detection* (CPPIED). To our knowledge, the only prior work to the one we presented in [Morin et al., 2013b] is that of [Drabovich, 2008]. He presented the advantages and the disadvantages of existing coverage path planning approaches and concluded that none of them was suitable for the imperfect extended detection in the context of underwater minesweeping operations. We further formalized the model proposed in [Drabovich, 2008] and proposed the CPPIED formulation in [Morin et al., 2013b].

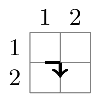
As the CPPIED problem deals with a required coverage in each cell, we also need to distinguish it from the *multirobot-controlled frequency coverage* (MRCFC) problem [Cannata and Sgorbissa, 2011]. The two problems are similar in that many visits will be needed in each location (cell). However, the goal of the former is to achieve the required coverage in each cell



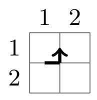
(a) A move from cell $(1, 1)$ to $(1, 2)$



(b) A move from cell $(1, 1)$ to $(2, 1)$



(c) A right turn from cell $(1, 1)$ to $(2, 1)$



(d) A left turn from cell $(1, 1)$ to $(1, 1)$

Figure 3.2: Feasible moves on a uniform grid

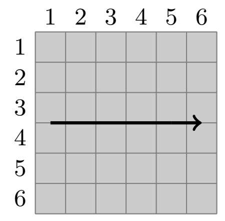
whereas the goal of the latter is to be as close as possible to the prescribed relative frequency of visits.

The only existing approach to the novel CPPIED is known as the *heterogeneous coverage path planning* (HCPP) algorithm [Drabovich, 2008]. Since practical instances of the CPPIED involve large grid environments (the published experimental instances contain more than 21 thousand cells) the HCPP algorithm is a heuristic. However, it is highly instance dependent as it must be fine-tuned in a trial and error fashion over each instance. We propose the *dynamic programming sweeper* (DpSweeper) algorithm, a different heuristic approach which requires minimal tuning. The experimental results on problem instances from the literature show that DpSweeper outperforms HCPP.

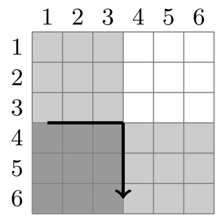
This chapter is organized as follows. We formalize the CPPIED in the context of underwater minesweeping operations in Section 3.1. Section 3.2 describes our novel hybrid algorithm based on *dynamic programming* (DP) and on a *traveling salesman problem* (TSP) reduction. We compare experimental results to the ones of the HCPP algorithm in Subsection 3.2.4. We conclude in Section 3.3.

3.1 The Coverage Path Planning Problem with Imperfect Extended Detection

Let $\mathcal{T}$ be the set of seabed types, for example, flat or ripple sand. The ocean is represented by an $m \times n$ matrix $\mathbf{O}$ such that $\mathbf{O}_{ij} \in \mathcal{T}$ is the seabed type in the cell (i, j). The cell $(1, 1)$ is located in the upper left corner of the grid. The position of the robot is defined by $((i, j), dir)$ where (i, j) is a grid cell and $dir \in \{\text{north}, \text{south}, \text{east}, \text{west}\}$ is the direction the robot is facing. The robot moves on the grid lines between the cells. A robot pointing north or south in cell (i, j) is located in the middle of the vertical line between cells (i, j) and $(i, j + 1)$. A robot pointing east or west in cell (i, j) is located in the middle of the horizontal line between cells (i, j) and $(i + 1, j)$. The robot moves forward one cell at a time (see Figures 3.2(a) and 3.2(b)). It cannot stop, backup, nor turn around on the spot. However, 90° turns while moving forward are allowed (see Figures 3.2(c) and 3.2(d)). The robot scans $2r^{\max}$ cells: $r^{\max}$



(a) A path from cell $(3, 1)$ to cell $(3, 6)$ with $r^{\max} = 3$



(b) A path from cell $(3, 1)$ to cell $(6, 3)$ with $r^{\max} = 3$

Figure 3.3: Scans on a uniform seabed; light gray shaded cells are scanned once, and dark gray shaded cells are scanned twice.

on its left, $r^{\max}$ on its right. No diagonal scanning occurs while the robot is turning. However, some cells may be scanned a second time just after the turn. Two possible paths and their set of scanned cells are shown on Figure 3.3 for range $r^{\max} = 3$.

The conditional detection probability of a sensor scan in a cell (i, j) (given that a mine is present) is a function $p^{\text{scan}} : \mathbb{N}^{+} \times \mathcal{T} \to [0.0, 1.0]$ of the distance $d(x, y, i, j)$, in number of cells, between the current robot's cell (x, y) and the scanned cell (i, j), and of the seabed type $\mathbf{O}_{ij}$ of the scanned cell (i, j). We represent the p^{scan} function as a $|\mathcal{T}| \times r^{\max}$ matrix. For instance, with $\mathcal{T} = \{\text{flat (f), ripples (r), complex (c)}\}$ and $r^{\max} = 3$, the sensor's conditional detection probabilities are expressed as:

$$p^{\text{scan}} = \begin{bmatrix} p^{\text{scan}}(1, \text{c}) & p^{\text{scan}}(2, \text{c}) & p^{\text{scan}}(3, \text{c}) \\ p^{\text{scan}}(1, \text{r}) & p^{\text{scan}}(2, \text{r}) & p^{\text{scan}}(3, \text{r}) \\ p^{\text{scan}}(1, \text{f}) & p^{\text{scan}}(2, \text{f}) & p^{\text{scan}}(3, \text{f}) \end{bmatrix}. \tag{3.1}$$

The current coverage map of a grid environment is represented by a $m \times n$ matrix $\mathbf{C}$ where $\mathbf{C}_{ij}$ is the achieved conditional detection probability in the cell (i, j). The initial coverage map is the null matrix. After a first scan of cell (i, j) from cell (x, y), the coverage in (i, j) is $\mathbf{C}_{ij} = p^{\text{scan}}(d(x, y, i, j), \mathbf{O}_{ij})$. The conditional probability of non-detection (i.e., the miss probability, or the non-coverage) is $1 - \mathbf{C}_{ij} = 1 - p^{\text{scan}}(d(x, y, i, j), \mathbf{O}_{ij})$. Given independent detections and following a scan in cell (i, j) from cell (x', y'), the non-coverage in cell (i, j) is $1 - \mathbf{C}'_{ij} = (1 - \mathbf{C}_{ij})(1 - p^{\text{scan}}(d(x', y', i, j)), \mathbf{O}_{ij})$. Therefore, the updated coverage of cell (i, j) is $\mathbf{C}'_{ij} = 1 - (1 - \mathbf{C}_{ij})(1 - p^{\text{scan}}(d(x', y', i, j)), \mathbf{O}_{ij})$ leading to the update equation (3.2).

$$\mathbf{C}'_{ij} \coloneqq \mathbf{C}_{ij} + (1 - \mathbf{C}_{ij}) p^{\text{scan}}(d(x', y', i, j), \mathbf{O}_{ij}). \tag{3.2}$$

That is, a mine may have already been detected in (i, j) with a probability $\mathbf{C}_{ij}$, or it had not been detected before with a probability $(1 - \mathbf{C}_{ij})$ and it will be detected now from cell (x', y') with a conditional probability $p^{\text{scan}}(d(x', y', i, j), \mathbf{O}_{ij})$.

The required coverage is represented by an $m \times n$ matrix $\mathbf{D}$ such that $\mathbf{D}_{ij}$ represents the required minimum coverage that must be attained in cell (i, j). The required coverage is

	1	2	3	4	5	6
1	c .5	r .5	f .5	f .5	f .5	f .5
2	c .5	r .5	f .7	f .7	r .5	r .5
3	c .8	r .8	r .8	r .8	r .8	r .8
4	r .8	r .8	r .8	r .8	r .8	r .8
5	f .5	f .5	f .7	f .7	f .5	f .5
6	f .5	f .5	f .5	f .5	f .5	f .5

Figure 3.4: A CPPIED problem instance with $\mathcal{T} = \{\text{flat (f), ripples (r), complex (c)}\}$, and $r^{\max} = 1$; dark gray shaded cells are complex seabed, medium gray shaded cells are ripples seabed, light gray shaded cells are flat seabed. The required coverage is displayed in each cell.

achieved when $\mathbf{D} \leq \mathbf{C}$. The required coverage is *homogeneous* when $\mathbf{D}_{ij}$ is equal for all rows i and columns j. Otherwise, it is *heterogeneous*. Instances from [Drabovich, 2008] feature a homogeneous required coverage (i.e., uniform values across $\mathbf{D}$) and a heterogeneous seabed (i.e., $|\mathcal{T}| > 1$).

A CPPIED problem instance is defined as a tuple:

$$\left(\mathcal{T}, \mathbf{O}, \mathbf{D}, p^{\text{scan}}, (i^{\text{init}}, j^{\text{init}})\right), \tag{3.3}$$

where $(i^{\text{init}}, j^{\text{init}})$ is the initial location cell of the robot. The initial direction parameter is not necessary because it may be inferred directly from the first segment of the robot's path. A solution is a robot's path P along with the directions of the robot at each waypoint. The direction of the robot along that path influences the set of visible cells. A feasible solution respects the robot's physical constraints while enabling it to achieve the required coverage. This solution is entirely defined by the sequence of moves of the robot M from its starting position. The problem, as formulated with two objectives (i.e., to minimize the distance and to minimize the turns), is indeed a multi-objective optimization problem. Depending on the units we use, these objectives may be incommensurable. In this case, a lexicographic ordering of the criteria can be used for optimization purposes. This is the case in [Drabovich, 2008] and [Morin et al., 2013b] where an optimal solution minimizes the distance first (in number of moves) and then the number of turns.

Example 3.1.1 (Surveying the seabed to detect naval mines)**.** Suppose that we want to solve the CPPIED instances presented on Figure 3.4. We assume a maximal visibility range of $r^{\max} = 1$. The heterogeneous seabed of this environment is made of 3 different seabed types:

$$\mathcal{T} = \{\text{flat(f)}, \text{ripples(r)}, \text{complex(c)}\}. \tag{3.4}$$

The detection model, i.e., the p^{scan} function, is assumed to be:

$$p^{\text{scan}} = \begin{bmatrix} p^{\text{scan}}(1, \text{c}) \\ p^{\text{scan}}(1, \text{r}) \\ p^{\text{scan}}(1, \text{f}) \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.8 \\ 0.99 \end{bmatrix}. \tag{3.5}$$

The seabed matrix is:

$$\mathbf{O} = \begin{bmatrix} \text{c} & \text{r} & \text{f} & \text{f} & \text{f} & \text{f} \\ \text{c} & \text{r} & \text{f} & \text{f} & \text{r} & \text{r} \\ \text{c} & \text{r} & \text{r} & \text{r} & \text{r} & \text{r} \\ \text{r} & \text{r} & \text{r} & \text{r} & \text{r} & \text{r} \\ \text{f} & \text{f} & \text{f} & \text{f} & \text{f} & \text{f} \\ \text{f} & \text{f} & \text{f} & \text{f} & \text{f} & \text{f} \end{bmatrix}. \tag{3.6}$$

The required coverage matrix is:

$$\mathbf{D} = \begin{bmatrix} .5 & .5 & .5 & .5 & .5 & .5 \\ .5 & .5 & .7 & .7 & .5 & .5 \\ .8 & .8 & .8 & .8 & .8 & .8 \\ .8 & .8 & .8 & .8 & .8 & .8 \\ .5 & .5 & .7 & .7 & .5 & .5 \\ .5 & .5 & .5 & .5 & .5 & .5 \end{bmatrix}. \tag{3.7}$$

For the sake of simplifying the discussion, we suppose that $(i^{\text{init}}, j^{\text{init}})$ is just outside the grid in the upper left corner, i.e., the robot starts in cell $(0, 0)$. This allows the robot to travel on the line before the first row or on the line before the first column on its first path segment.

Starting from cell $(0, 0)$, a solution to this CPPIED instance would be entirely defined by the following sequence of moves:

$$\begin{aligned} M = (\quad & \text{east}, & \text{east}, & \ \ldots, & \text{east}, & \ \text{south}, & \text{south}, \\ & \text{west}, & \text{west}, & \ \ldots, & \text{west}, & \ \text{south}, & \text{south}, \\ & \text{east}, & \text{east}, & \ \ldots, & \text{east}, & \ \text{south}, & \text{south}, \\ & \text{west}, & \text{west}, & \ \ldots, & \text{west}\,). \end{aligned} \tag{3.8}$$

The first direction is the first robot's move from cell $(i^{\text{init}}, j^{\text{init}})$, the second is its second move and so on. Using cell $(i^{\text{init}}, j^{\text{init}})$ and the first move in M, it is easy to infer the best starting direction of the robot. It is also easy to complete the robot's path using its sequence of moves. Figure 3.5 presents the inferred robot's path. The achieved coverage for the entire path is:

$$\mathbf{C} = \begin{bmatrix} .7 & .99 & .99 & .99 & .99 & .9999 \\ .7 & .8 & .99 & .99 & .8 & .96 \\ .91 & .8 & .8 & .8 & .8 & .8 \\ .96 & .8 & .8 & .8 & .8 & .8 \\ .99 & .99 & .99 & .99 & .99 & .9999 \\ .99 & .99 & .99 & .99 & .99 & .9999 \end{bmatrix}. \tag{3.9}$$

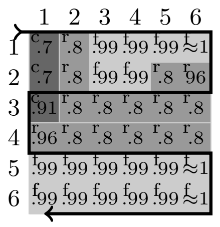

1 2 3 4 5 6
1 c .7 r .8 f .99 f .99 f .99 f ≈1
2 c .7 r .8 f .99 f .99 r .8 r .96
3 c .91 r .8 r .8 r .8 r .8 r .8
4 r .96 r .8 r .8 r .8 r .8 r .8
5 f .99 f .99 f .99 f .99 f .99 f ≈1
6 f .99 f .99 f .99 f .99 f .99 f ≈1

Figure 3.5: A tiny example of a CPPIED solution to the instance of Figure 3.4; dark gray shaded cells are complex seabed, medium gray shaded cells are ripples seabed, light gray shaded cells are flat seabed. The achieved coverage after the path's completion is displayed in each cell.

This solution is feasible since $\mathbf{D} \leq \mathbf{C}$ (and since it is within the robot's accessibility constraints). The path has 30 moves and 6 turns. It is optimal. $\triangleleft$

3.2 The Dynamic Programming Sweeper Algorithm

The goal of the proposed algorithm is to define, in lexicographic order, a feasible shortest path (the first objective), consisting of segments, that also minimizes the number of turns (the second objective). A segment is a set of horizontally or vertically adjacent cells that does not contain turns. The algorithm is composed of two main phases. In the first phase, we construct a set $\mathcal{S}$ of disconnected segments such that a robot traveling along all these segments will achieve the required coverage $\mathbf{D}$. In the second phase, we use a TSP reduction to optimally connect the segments obtained in phase 1 and form the desired path P. The segments order of visit is enough to reconstruct a complete solution M encoded as a sequence of moves from the starting position of the robot. Algorithm 1 outlines the DpSweeper algorithm. The first phase is detailed in Section 3.2.1. The second phase is detailed in Section 3.2.2.

3.2.1 Coverage with Segment Sets Using Dynamic Programming

Algorihtm 1 first initializes the robot's path M to the empty path, the set of segments $\mathcal{S}$ to the empty set, and the current coverage matrix $\mathbf{C}$ to the null matrix. Then, at each iteration, it improves coverage by adding a set $\mathcal{H}^*$ of horizontal segments or a set $\mathcal{V}^*$ of vertical segments to $\mathcal{S}$. Some difficulties in constructing a segment set $\mathcal{S}$ arise due to the extended sensor's range since multiple detections in a given cell may arise from two different segments or more. To overcome this difficulty, we impose, at each iteration, a $2r^{\max}$ spacing constraint within the set $\mathcal{H}^*$ and the set $\mathcal{V}^*$. Therefore, the rewards of the parallel segments are independent, which allows us to use polynomial time dynamic programming algorithms to compute both the reward of a robot traveling on a segment, and the sets of segments $\mathcal{H}^*$ and $\mathcal{V}^*$.

In order to compute $\mathbf{H}_{ij}$ the horizontal gain in cell (i, j), let $p^{\mathrm{C}}(i, j, k)$ be the updated prob-

```
Function DpSweeper(CPPIED)
    Input: A CPPIED instance: CPPIED = $(\mathcal{T}, \mathbf{O}, \mathbf{D}, p^{\text{scan}}, (i^{\text{init}}, j^{\text{init}}))$.
    Output: A feasible sequence of moves: $M$.
    Initialize $M$ to the empty path, $\mathcal{S}$ to the empty set, and $\mathbf{C}$ to the null matrix;
    // Phase 1 (See Section 3.2.1)
    while $\mathbf{C} < \mathbf{D}$ do
        Compute $\mathcal{H}^*$ and $\mathcal{V}^*$;
        $\mathbf{C}^{\mathcal{H}^*} \leftarrow$ UpdateCoverage ($\mathcal{H}^*$, $\mathcal{S} \cup \mathcal{H}^*$, $\mathbf{C}$, CPPIED);
        $\mathbf{C}^{\mathcal{V}^*} \leftarrow$ UpdateCoverage ($\mathcal{V}^*$, $\mathcal{S} \cup \mathcal{V}^*$, $\mathbf{C}$, CPPIED);
        $\mathcal{K} \leftarrow$ ChooseSet ($\mathcal{H}^*$, $\mathcal{V}^*$, $\mathbf{C}^{\mathcal{H}^*}$, $\mathbf{C}^{\mathcal{V}^*}$, CPPIED);
        $\mathbf{C} \leftarrow \mathbf{C}^{\mathcal{K}}$;
        $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{K}$;
    // Phase 2 (See Section 3.2.2)
    TSP $\leftarrow$ ReduceToTsp ($\mathcal{S}$, CPPIED);
    $TOUR \leftarrow$ SolveTsp (TSP);
    $M \leftarrow$ RetrievePath ($TOUR$, CPPIED);
    return $M$;
```

Algorithm 1: The DpSweeper algorithm

ability of detection when a cell (i, j) is scanned from a distance of k cells:

$$p^{\text{C}}(i, j, k) = \mathbf{C}_{ij} + (1 - \mathbf{C}_{ij})p^{\text{scan}}(k, \mathbf{O}_{ij}). \tag{3.10}$$

The gain obtained by scanning cell (i, j) from distance k can be defined as:

$$g(i, j, k) = \begin{cases} \min\left\{\mathbf{D}_{ij}, p^{\text{C}}(i, j, k)\right\} - \mathbf{C}_{ij} & \mathbf{C}_{ij} < \mathbf{D}_{ij}; \\ -\lambda & \mathbf{C}_{ij} \geq \mathbf{D}_{ij}. \end{cases} \tag{3.11}$$

If cell (i, j) is not already covered, i.e., $\mathbf{C}_{ij} < \mathbf{D}_{ij}$, the gain equals the increment in detection probability in (i, j) up to the required coverage value, i.e., $\min\left\{\mathbf{D}_{ij}, p^{\text{C}}(i, j, k)\right\} - \mathbf{C}_{ij}$. Otherwise, cell (i, j) is covered, i.e., $\mathbf{C}_{ij} \geq \mathbf{D}_{ij}$, and we impose a small penalty λ for overcoverage. Let $G(i, j)$ be the sum of the gains, in probability of detection, in the cells within the range of the robot when it is positioned horizontally in cell (i, j):

$$G(i, j) = \sum_{k=1}^{r^{\max}} g(i - k + 1, j, k) + \sum_{k=1}^{r^{\max}} g(i + k, j, k). \tag{3.12}$$

The penalty λ, attributed to a scan in a visible cell that is already covered, is chosen small enough to avoid interfering (up to a desired decimal precision) with the gains obtained in other visible cells. Therefore, the sum of the gains when the robot is positioned horizontally in cell (i, j) is strictly positive as soon as there is at least one visible uncovered cell from that position. The horizontal gain of a cell (i, j), $\mathbf{H}_{ij}$ should reflect the fact that we wish to find the shortest possible segment with the highest possible coverage. It is thus convenient to end the segment before the first cell (i, j) for which $G(i, j)$ is less than or equal to zero. Therefore, the horizontal gain of a cell (i, j), $\mathbf{H}_{ij}$, is defined as follows:

$$\mathbf{H}_{ij} = \begin{cases} G(i,j) & G(i,j) > 0; \\ -\infty & G(i,j) \leq 0. \end{cases} \tag{3.13}$$

The endpoints a and b that define the segment h_i such that the sum of its gains is maximal are identified for each row i as follows:

$$K(h_i) = \max_{a,b} \sum_{j=a}^{b} \mathbf{H}_{ij}. \tag{3.14}$$

This problem, called the *Maximum subarray* problem, is solvable in linear time using Kadane's algorithm [Bentley, 1984]. Kadane's algorithm is a simple example of dynamic programming.

The optimal horizontal set $\mathcal{H}^*$ is the subset of segments that maximizes the sum of the horizontal gains over the rows subject to a $2r^{\max}$ spacing constraint:

$$\max_{I \subset \{1..m\} | i,i' \in I \Rightarrow |i-i'| \geq 2r^{\max}} \sum_{i \in I} K(h_i). \tag{3.15}$$

We compute the maximal horizontal gain under a $2r^{\max}$ spacing constraint using the following recurrence relation:

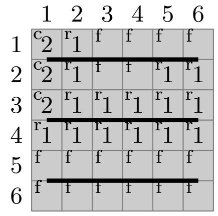
$$\mathbf{h^*}_i = \begin{cases} 0 & i < 1; \\ \max\left\{\mathbf{h^*}_{i-1}, K(h_i) + \mathbf{h^*}_{i-2r^{\max}}\right\} & 1 < i \leq m, \end{cases} \tag{3.16}$$

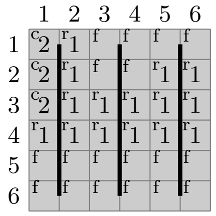
where $\mathbf{h^*}_i$ is the maximum gain achieved by a robot on the first i segments. Following a dynamic programming approach, vector $\mathbf{h}^*$ is stored in memory to exploit the optimal substructure property (see Section 1.2.3). Just as we did for the optimal tour in our dynamic programming TSP example of Section 1.2.3, the optimal horizontal set $\mathcal{H}^*$ is computed by backtracking in $\mathbf{h}^*$. With this technique, choosing an optimal set of segments under a $2r^{\max}$ spacing constraint is done in polynomial time. The set $\mathcal{V}^*$ is computed in the same fashion following a map rotation of 90°. Algorithm 1 computes the updated coverage matrices $\mathbf{C}^{\mathcal{H}^*}$ and $\mathbf{C}^{\mathcal{V}^*}$ for sets $\mathcal{H}^*$ and $\mathcal{V}^*$ using Equation (3.2) (function UpdateCoverage). The set among $\mathcal{H}^*$ or $\mathcal{V}^*$ providing the highest coverage is added to $\mathcal{S}$. Ties are broken using the set with the smallest cardinality (function ChooseSet). Note that there is no spacing constraint in $\mathcal{S}$.

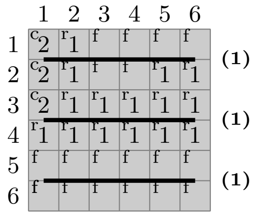
Example 3.2.1 (Constructing the segment set $\mathcal{S}$)**.** We now present an example for choosing a segment set $\mathcal{S}$ by using our method on a 6×6 grid with $r^{\max} = 1$. The set $\mathcal{S}$ is initialized to the empty set, and the current coverage matrix is initialize to zero, i.e., $\mathbf{C} = 0$. We represent the initial grid on Figure 3.6. For clarity purposes, we indicate, in each cell of the grid, the number of robot scans needed to achieve the required coverage $\mathbf{D}$ instead of the current coverage probability matrix. This simplification is possible since $r = 1$. At iteration 1 (see

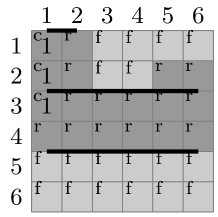
	1	2	3	4	5	6
1	$^{c}3$	$^{r}2$	$^{f}1$	$^{f}1$	$^{f}1$	$^{f}1$
2	$^{c}3$	$^{r}2$	$^{f}1$	$^{f}1$	$^{r}2$	$^{r}2$
3	$^{c}3$	$^{r}2$	$^{r}2$	$^{r}2$	$^{r}2$	$^{r}2$
4	$^{r}2$	$^{r}2$	$^{r}2$	$^{r}2$	$^{r}2$	$^{r}2$
5	$^{f}1$	$^{f}1$	$^{f}1$	$^{f}1$	$^{f}1$	$^{f}1$
6	$^{f}1$	$^{f}1$	$^{f}1$	$^{f}1$	$^{f}1$	$^{f}1$

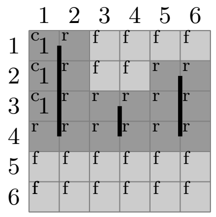
Figure 3.6: An example of the initial number of scans required to cover a simple seabed map with $\mathcal{T} = \{\text{flat (f), ripples (r), complex (c)}\}$, and $r^{\max} = 1$

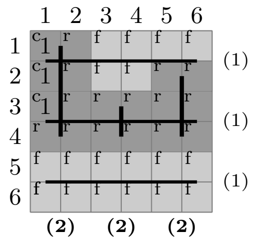

(a) Iteration 1: a set $\mathcal{H}^*$ of horizontal segments is generated.

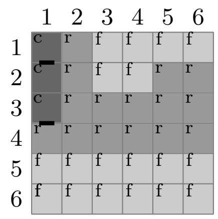

(b) Iteration 1: a set $\mathcal{V}^*$ of vertical segments is generated.

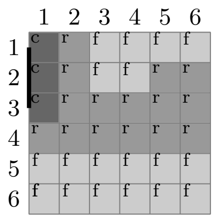

(c) Iteration 1: a horizontal segment set $\mathcal{H}^*$ (labeled (1)), is added to $\mathcal{S}$.


(d) Iteration 2: a set $\mathcal{H}^*$ of horizontal segments is generated.


(e) Iteration 2: a set $\mathcal{V}^*$ of vertical segments is generated.


(f) Iteration 2: a vertical segment set $\mathcal{V}^*$ (labeled (2)) is added to $\mathcal{S}$.


(g) Iteration 3: a set $\mathcal{H}^*$ of horizontal segments is generated.


(h) Iteration 3: a set $\mathcal{V}^*$ of vertical segments is generated.

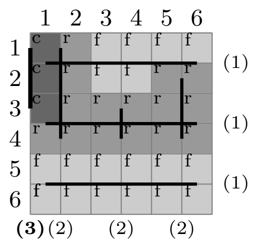

(i) Iteration 3: a vertical segment set $\mathcal{V}^*$ (labeled (3)) is added to $\mathcal{S}$.

Figure 3.7: An example of the construction of the disconnected segment set on a map with $\mathcal{T} = \{\text{flat (f), ripples (r), complex (c)}\}$, and $r^{\max} = 1$; dark gray shaded cells were scanned three times, medium gray shaded cells were scanned twice, light gray shaded cells were scanned once.

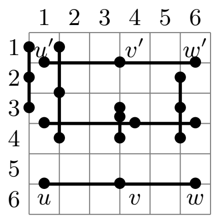
Figures 3.7(a) to 3.7(c)), the procedure generates a first set $\mathcal{H}^*$ of horizontal segments and a first set $\mathcal{V}^*$ of vertical segments. $\mathcal{H}^*$ is then added to $\mathcal{S}$ ($\mathcal{V}^*$ is equivalent in terms of gain, in this case). At iteration 2 (see Figures 3.7(d) to 3.7(f)), the updated current coverage matrix $\mathbf{C}$ is used to compute two new sets $\mathcal{H}^*$ and $\mathcal{V}^*$. The algorithm adds $\mathcal{V}^*$ to $\mathcal{S}$ since it achieves a higher gain. This choice is justified as follows. There are, when using the horizontal segment set $\mathcal{H}^*$, multiple cells that are already covered leading to small penalties. We also notice that the total length of the segments in $\mathcal{H}^*$ is greater than the total length in the segments of $\mathcal{V}^*$. At iteration 3 (see Figures 3.7(g) to 3.7(i)), the algorithm chooses $\mathcal{V}^*$ (a single segment) and adds it to $\mathcal{S}$. This choice is justified as follows. The horizontal segment positioned in cell $(3, 1)$ of the horizontal segment set $\mathcal{H}^*$ leads to a small penalty for overcoverage. Furthermore, the cardinality of $\mathcal{H}^*$ is greater than the cardinality of $\mathcal{V}^*$. We see, on Figure 3.7(i), the completed set of disconnected $\mathcal{S}$ that will be used during the second phase of the algorithm. $\triangleleft$

3.2.2 Linking the Segments Using a Traveling Salesman Tour

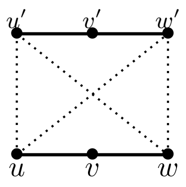
In this phase, a connected path P including all the segments in $\mathcal{S}$ is constructed using a TSP reduction (function ReduceToTsp of Algorithm 1). We first construct, using the segment set $\mathcal{S}$ built during the first phase of the DpSweeper algorithm, a graph G on which to plan a TSP cycle. Let $\mathcal{V}(G)$ be the set of nodes of a graph G, $\mathcal{E}(G)$ be its set of edges, and $c(e)$ be a cost function on $\mathcal{E}(G)$. The goal of the TSP is to find a cycle of minimal cost that visits each node once. The original TSP formulation involves complete graphs whereas ours deals with arbitrary graphs. An arbitrary graph G is easily transformed into a complete graph for the TSP by adding an edge (u, v) of infinite cost to G for each pair of vertices $u, v \in \mathcal{V}(G)$ such that (u, v) is a missing edge of G, i.e., $(u, v) \notin \mathcal{E}(G)$. The reduction goes as follows.

For each segment s_i in $\mathcal{S}$ with endpoints u_i and w_i, we create three nodes in $\mathcal{V}(G)$: u_i, w_i, and v_i and two edges (u_i, v_i) and (v_i, w_i). The node v_i is a dummy node used to force the algorithm to travel over the segment between u_i and w_i. The cost of these two edges is null. For every pair of segments s_i and s_j in $\mathcal{S}$, we create four edges: (u_i, u_j), (u_i, w_j), (w_i, u_j), and (w_i, w_j). The cost of these edges corresponds to the shortest distance in the number of robot's moves needed to connect the endpoints. Finally, let u^s be the starting position of the robot. We create a source node u^s connected to all endpoints of the segments with distances given by the shortest distance in number of moves. We create a sink node w^s connected to all segments' endpoints with a null distance. We create a dummy node v^s that is connected to u^s and w^s by two edges of null cost.

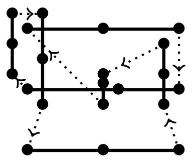
Solving the TSP consists of finding a cycle of minimum cost that passes through each node exactly once. Note that such a cycle would have to visit each node v_i once. Since this node is only connected to the two endpoints u_i, w_i, it forces the cycle to enter by one endpoint of the segment and to leave by the other endpoint. The same principle applies for the node v^s that is only connected to the source node u^s and the sink node w^s. Therefore, a cycle starts at

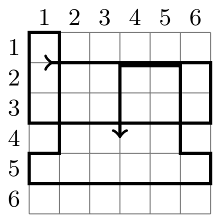


(a) Step 1: generate the graph's nodes.



(b) Step 2: generate the graph's edges (dotted lines).


(c) Step 3: find an optimal cycle (dotted lines).



(d) Step 4: reconstruct the path.

Figure 3.8: TSP reduction and path reconstruction for the segments set $\mathcal{S}$ of Figure 3.7

node w^s, visits all the segments, and returns to w^s through v^s. Removing the node w^s and v^s from the cycle yields the solution path P we are looking for. To find the lowest cost cycle, and therefore the solution path, we chose the Concorde solver [Applegate et al., 2011] (function SolveTsp of Algorithm 1). We chose to use Concorde instead of a TSP heuristic since it is the state of the art. Concorde solves most TSPLIB [Reinelt, 1991] instances (sometimes with more than 2 thousand nodes) within a few minutes. Although Concorde may consume more time than a heuristic, it finds the optimal tour to all our instances within a few seconds. The solution to the problem, in terms of sequence of moves M, follows directly (function Retrieve Path of Algorithm 1).

Example 3.2.2 (Constructing the path)**.** Figure 3.8 shows a TSP reduction and a path reconstruction example on the segments set $\mathcal{S}$ of Figure 3.7. First, the reduction process generates the nodes represented by dots on Figure 3.8(a). The robot starts in cell $(1, 1)$. Therefore, the source node u^s is positioned in $(1, 1)$, the sink node w^s and the dummy node v^s are positioned at the same place. Second, it links the segments endpoints by creating edges. Figure 3.8(b) shows the edges added to the graph G for segments $s = (u, v, w)$ and $s' = (u', v', w')$. Third, on Figure 3.8(c), the Concorde solver finds an optimal cycle that starts from the source and goes through all the nodes before returning to the source node u^s by the sink node w^s then the dummy node v^s. Finally, on Figure 3.8(d), the robot's path is reconstructed. As shown, there are no links between the last robot's position $(4, 3)$ and its initial position in $(1, 1)$: We simply ignore the sink and the dummy nodes during path reconstruction. ◁

Table 3.1: Problem instances published in [Drabovich, 2008]

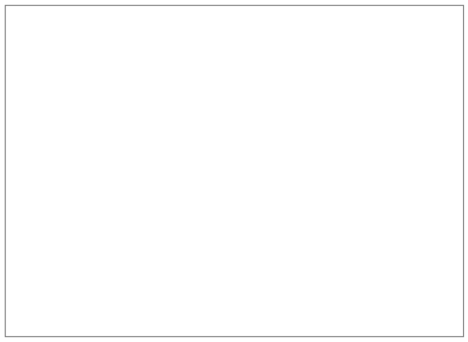
No	Ocean seabed $\mathbf{O}$	Required coverage $\mathbf{D}_{ij}$ $(\forall i, j)$	Detection probrability p^{scan}
1	Flat seabed Figure 3.9(a)	0.9	$\begin{bmatrix} 0.6 & 0.65 & 0.62 \\ 0.7 & 0.85 & 0.8 \\ 0.99 & 0.99 & 0.99 \end{bmatrix}$
2	Rectangular patch Figure 3.9(b)	0.9	$\begin{bmatrix} 0.7 & 0.8 & 0.75 \\ 0.99 & 0.99 & 0.99 \\ 0.91 & 0.95 & 0.92 \end{bmatrix}$
3	3 rectangular patches Figure 3.9(c)	0.9	$\begin{bmatrix} 0.8 & 0.8 & 0.8 \\ 0.8 & 0.8 & 0.8 \\ 0.91 & 0.91 & 0.91 \end{bmatrix}$
4	Circular patch Figure 3.9(d)	0.75	$\begin{bmatrix} 0.51 & 0.51 & 0.51 \\ 0.8 & 0.8 & 0.8 \\ 0.91 & 0.91 & 0.91 \end{bmatrix}$
5	Fragmented circular patch Figure 3.9(e)	0.75	$\begin{bmatrix} 0.51 & 0.51 & 0.51 \\ 0.8 & 0.8 & 0.8 \\ 0.99 & 0.99 & 0.99 \end{bmatrix}$
6	Real map Figure 3.9(f)	0.9	$\begin{bmatrix} 0.6 & 0.6 & 0.6 \\ 0.8 & 0.8 & 0.8 \\ 0.91 & 0.91 & 0.91 \end{bmatrix}$
7	Random map Figure 3.9(g)	0.85	$\begin{bmatrix} 0.6 & 0.6 & 0.6 \\ 0.8 & 0.8 & 0.8 \\ 0.91 & 0.91 & 0.91 \end{bmatrix}$

3.2.3 Experiments

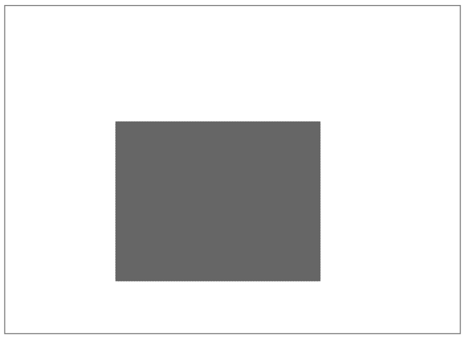
We present the results of our DpSweeper algorithm, implemented in C++, obtained on an Intel(R) Core(TM) i7-Q740 CPU with 8 GB of RAM. The seabed maps $\mathbf{O}$ of the seven problem instances used as a benchmark (see Figure 3.9) contain more than 21 thousand cells. Each cell is made of a flat (f) seabed, of sand ripples (r), or of complex (c) seabed. The first instance, shown in Figure 3.9(a), has a flat seabed: a lawnmower (boustrophedon) pattern consisting of parallel segments is sufficient to cover it. The second, shown in Figure 3.9(b), is made of a flat seabed with a rectangular complex seabed patch in the middle. The third, shown in Figure 3.9(c), has three complex seabed patches. The fourth, shown in Figure 3.9(d), has a circle of complex seabed in the middle. The fifth, shown in Figure 3.9(e), has a fragmented circle of complex seabed in the middle. The sixth, shown in Figure 3.9(f), is a real map containing three different seabed types. The seventh, shown in Figure 3.9(g), is a realistic randomly generated map containing three different seabed types. Table 3.1 summarizes the instances in increasing order of complexity. The robot starts in the top left corner of the grid. The required coverage matrix $\mathbf{D}$ is uniform, i.e., all cells have the same required coverage probability. We use a range $r^{\max} = 3$.

3.2.4 Results and Discussion

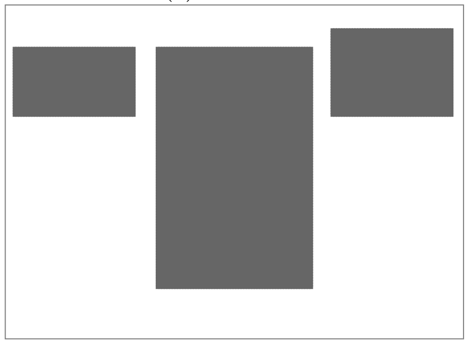
Table 3.2 presents the results of both the HCPP and the DpSweeper algorithms. The required coverage is attained in all cases. The first group of columns presents the results as published in [Drabovich, 2008]. The HCPP algorithm is a parameterized heuristic which requires choos-



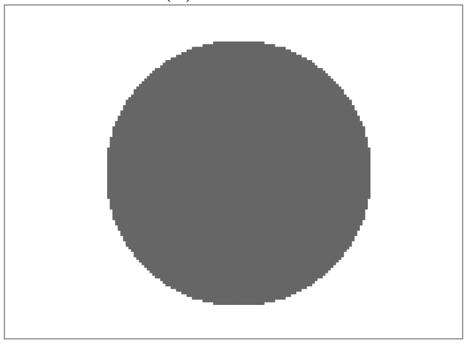
(a) Instance 1



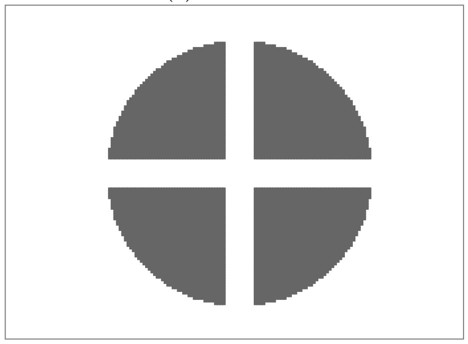
(b) Instance 2



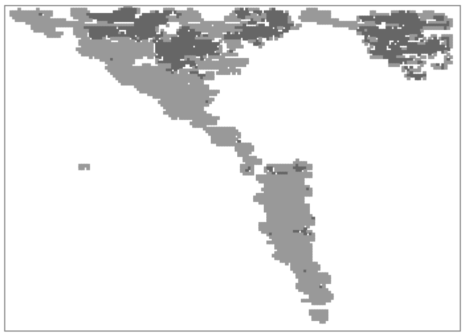
(c) Instance 3



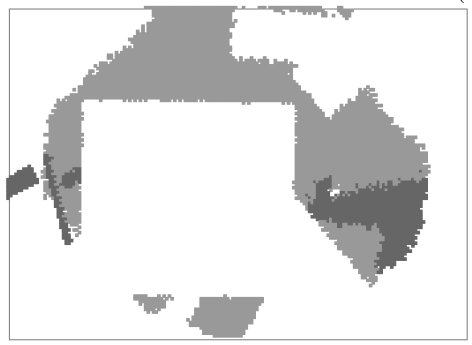
(d) Instance 4



(e) Instance 5



(f) Instance 6



(g) Instance 7

Figure 3.9: The maps of the instances published in [Drabovich, 2008]; dark gray shaded cells are complex seabed, light gray shaded cells are ripples seabed, white cells are flat seabed.

Table 3.2: Comparison of DpSweeper to HCPP [Drabovich, 2008]

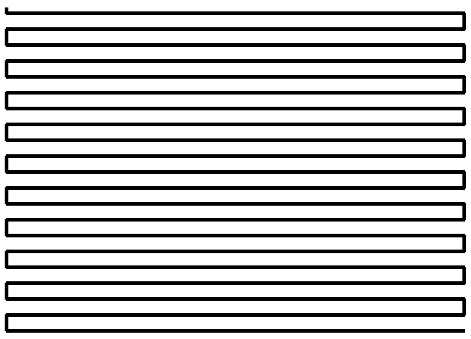
No	***HCPP***[†]			***DpSweeper***					
	Solution			Solution			TSP	Time (s)	
	Length	Turns	Lex. dom.[‡]	Length	Turns	Lex. dom.[‡]	nodes	TSP	Total
1	3777	$\underline{40}$	no	$\underline{3755}$	41	yes	66	< 1	< 1
2	$\underline{4599}$	$\underline{60}$	yes	4628	61	no	99	< 1	1
3	5556	$\underline{82}$	no	$\underline{5321}$	95	yes	144	< 1	1
4	5494	77	no	$\underline{5363}$	$\underline{75}$	yes	117	< 1	1
5	$\underline{5119}$	114	yes	5151	$\underline{107}$	no	162	1	2
6	6689	$\underline{136}$	no	$\underline{5681}$	270	yes	402	45	52
7	7141	$\underline{137}$	no	$\underline{5731}$	174	yes	273	12	15
Wins	2	5	2	5	2	5			

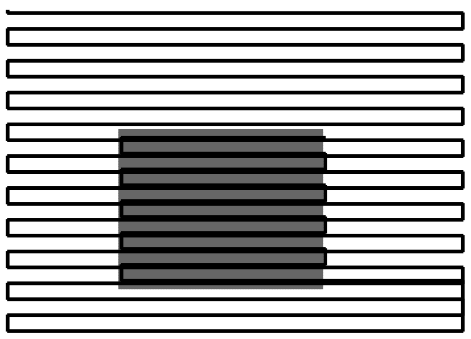
† The solving times of the HCPP method are not published.

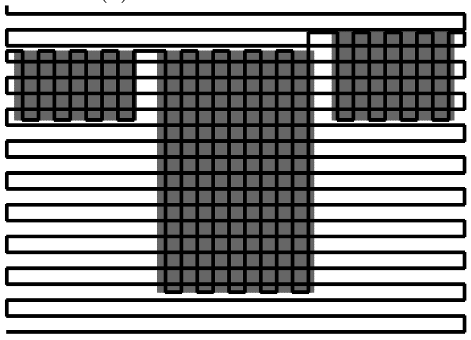
‡ Lexicographic dominance: Shorter is better, ties are broken on the number of turns.

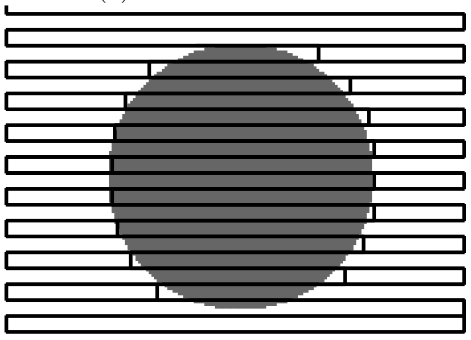
ing the set of parameters that performs best given the instance to solve. This configuration, as performed in [Drabovich, 2008], turns out to be done on a per instance fashion. For each instance, the set of parameters leading to the minimal length is kept. Ties, among parameter sets leading to the same path length, are broken according to the number of turns. This leads us to consider the following lexicographic order of the criteria for evaluation purposes (see the lexicographic dominance column of Table 3.2): minimize the length of the path in number of moves, then minimize the number of turns. We see that the DpSweeper algorithm outperforms the HCPP algorithm on the first criterion in all instances except for instances 2 and 5. Following a close inspection of the figures published in [Drabovich, 2008], we noticed that on the spot 180° turns were allowed. Our more restrictive assumptions (not allowing on the spot 180° turns), although not a DpSweeper limitation, are closer to the physical constraints of AUVs from the REMUS family and actually favor HCPP. This may explain the slightly longer path we found on instances 2 and 5. For the instances 6 and 7, our solutions have significantly more turns than HCPP. On these instances, generating more turns was necessary to diminish the path length of respectively more than 15% and 20% when compared to the HCPP. Favoring shorter paths is coherent with the lexicographic order of the minimized objectives. It is also worth adding that the DpSweeper algorithm solved instances 1 to 4 in one second or less and instance 5 in two seconds. The solving times of the most realistic instances (6 and 7) are within a minute. These times include both the segment generation process and the Concorde solver calls. It further confirms that the TSP instances resulting from the second phase of the DpSweeper algorithm are still within the reach of a TSP solver for the considered problem instances.

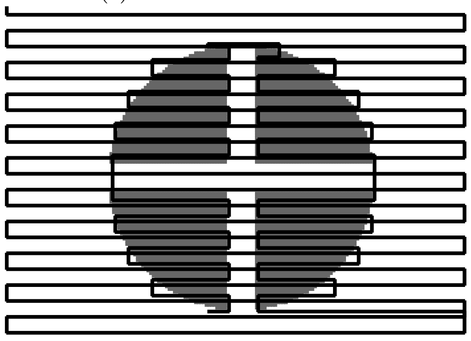
Figure 3.10 reports the solutions provided by our algorithm. The solution on instance 1 (see Figure 3.10(a)) is a lawnmower pattern. It is easily recognizable as an optimal solution (in terms of both objectives) on this flat seabed. On instance 2 (see Figure 3.10(b)), the only difficulty is to come back to cover the rectangular patch of complex seabed. On instance 3 (see Figure 3.10(c)), we see the tendency of the algorithm to generate a simple lawnmower pattern

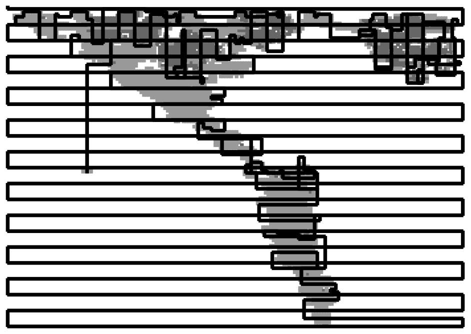

(a) Solution to instance 1


(b) Solution to instance 2


(c) Solution to instance 3


(d) Solution to instance 4


(e) Solution to instance 5


(f) Solution to instance 6

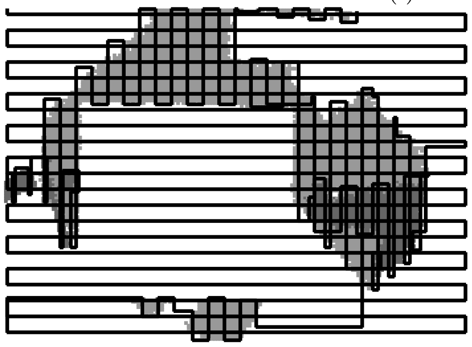

(g) Solution to instance 7

Figure 3.10: Solutions found by DpSweeper; complex, ripples, and flat seabed cells are respectively filled with dark gray, gray, and white.

Table 3.3: Comparison of DpSweeper to HCPP [Drabovich, 2008] as implemented in [Fruitet, 2013]

No	***HCPP′ with* 180°**†			***HCPP′ without* 180°**†			***DpSweeper***		
		Solution			Solution			Solution	
	Length	Turns	Lex. dom.‡	Length	Turns	Lex. dom.‡	Length	Turns	Lex. dom.‡
1	3755	41	—	—	—	—	3755	41	—
2	4628	61	—	—	—	—	4628	61	—
3	5538	83	no	5501	101	no	<u>5321</u>	95	yes
4	5503	79	no	5511	85	no	<u>5363</u>	<u>75</u>	yes
5	5374	109	no	5374	109	no	<u>5151</u>	<u>107</u>	yes
6	6178	<u>145</u>	no	6438	151	no	<u>5681</u>	270	yes
7	6545	<u>132</u>	no	6673	192	no	<u>5731</u>	174	yes
Wins	0	3	0	0	0	0	5	2	5

† The solving times are irrelevant since Matlab is an interpreted language.
‡ Lexicographic dominance: Shorter is better, ties are broken on the number of turns.

that minimizes both the path length and the number of turns. It starts with a horizontal lawnmower pattern. Then, on the third row, it follows a vertical lawnmower to cover the complex seabed patches. Finally, it comes back to its horizontal lawnmower pattern to cover the rest of the grid. We notice a similar behavior on instance 4 (see Figure 3.10(d)). The circle edge adds complexity to that instance since the algorithm is forced to adapt the length of the segments. The same is true for instance 5 (see Figure 3.10(e)). For instance 6 (see Figure 3.10(f)), the algorithm first plans a complete lawnmower pattern. Then, it goes up towards and over the closest ripples seabed and the complex patches. Finally, it passes over complex seabed patches again to achieve the required coverage, and then aims for the furthest ripples patches. It ends its course on the left hand side of the map. A similar behavior occurs in Figure 3.10(g).

Table 3.3 compares the DpSweeper algorithm to a novel Matlab [Mathworks, 2010] implementation of the HCPP algorithm [Fruitet, 2013].[1] We call this reimplementation HCPP′. On the spot 180° turns were allowed in a first version of HCPP to make sure that the results are of the same order as the ones obtained in [Drabovich, 2008]. In a second version of the algorithm, on the spot 180° turns were disallowed. Instances 1 and 2 were not evaluated for the HCPP′ algorithm without 180°. These environments are mainly used to validate the behavior of the algorithms. This comparison further confirms the prior results.

In addition to its superior results, the main advantage of DpSweeper over HCPP is that it is readily applicable to a problem instance. That is, contrary to HCPP, DpSweeper does not require lengthy instance-specific fine-tuning. Furthermore, the very short computational time of DpSweeper makes it possible to rapidly obtain a high quality path, an important characteristic for algorithms in practical contexts.

[1] The reimplementation of the HCPP algorithm is the work of Armand Fruitet, a work realized as part of an internship to study the problem in its bi-criteria form [Fruitet, 2013].

3.3 Concluding Remarks

We have presented a novel algorithm (DpSweeper) for coverage path planning using imperfect sensors with an extended detection range (CPPIED). The algorithm consists of two main phases. First, it greedily constructs a partial path made of segments to enforce the required coverage constraint. Then, it links the segments to create a path that is within the robot's physical constraints. The algorithm yields favorable results in a very short time compared to the literature. It is flexible and can be applied to complex seabed environments. In contrast with the only other algorithm that tackles the CPPIED problem, it does not require customized fine-tuning for individual environments. Even though the TSP problem is $\mathcal{NP}$-hard [Garey and Johnson, 1979], the instances resulting from our real and practical CPPIED problem instances are within the reach of the Concorde TSP solver that we used. TSP reductions are found in the literature on coverage problems (e.g., [Fang and Anstee, 2010]). However, most of these reductions use the TSP to find a sequence of large regions where they assume a fixed coverage pattern. We use the TSP directly to order a sequence of path components (segments) on the whole environment. Although our two phases approach proved to be efficient, it can lead to overcoverage. One aspect of the method is that the first phase does not account for the coverage occurring in the second phase. Further research could aim at improving the interaction between both phases to minimize overcoverage.

We described the CPPIED problem and the DpSweeper algorithm in terms of a coverage path planning problem for naval mines detection. They are, however, general. The use of the algorithm and of the CPPIED problem formalism in other obstacle-free environments for coverage is straightforward as long as the concept of independent paths is defined with respect to the modeled sensor, i.e., paths with non-overlapping detections must be identifiable. Obstacle-free environments appear, for instance, in aerial surveillance operations with *unmanned aerial vehicles* and other aircrafts. The adaptation of the algorithm to general environments with obstacles is also possible since it would simply require taking obstacles into account in the TSP reduction and during the generation of disconnected segments. The approach can also be generalized to other problems where the agent (may it be a robot) is allowed to turn at an angle different than 90°. This may be done by considering other grid types where the cells have a different shape, e.g., hexagonal grids. The approach can be generalized to various distant visibility constraints of the sensor as well. We considered lateral detections with respect to the position of the robot. Different types of visibility can be modeled and used within a method similar to DpSweeper by refining the concept of detection-independent paths (non overlapping detections) on graphs. This would allow for the use of fast dynamic programming algorithms for the first phase. Finally, the principle of partial coverage of a cell, that we called "an imperfect detection", is present in many applications. We can think of a partially cleaned floor (in robotic cleaning), a scanned area that was only partially visible due to the presence of obstacles (in surveillance), or, as we did, a detection probability in search operations.

Chapter 4

Search Path Planning with Markovian Motion

This chapter is based on our original work published in [Morin et al., 2012] and presented at the 18th International Conference on Principles and Practice of Constraint Programming (CP 2012).

We studied, in the previous chapter, the CPPIED problem, a coverage problem without any prior on the whereabouts of a search object. In this chapter, we move on to the search theory context. We now deal with a search path planning problem called the *optimal search path* (OSP) problem where the whereabouts are modeled as a probability distribution on the plausible location of the search object. We first present a CP model to solve the OSP problem with a Markovian motion of the object. As a second contribution, we refine the standard OSP problem objective function to obtain a tighter bound on the objective value of our CP model without discarding any solution. As a third contribution, we define the *total detection* (TD) heuristic which leads to a high objective value (i.e., *cos*) by directing the searcher towards vertices with high total probability of detecting the object. We use and describe the TD heuristic as a value selection heuristic for our CP model. In the first part of the experiments, we evaluate the novel objective function. In the second part of the experiments, we show how TD improves the results in short time. Experiments show that our model is competitive with published results from the search theory literature which features problem-specific B&B algorithms, e.g., [Lau et al., 2008].

We chose to tackle the OSP using CP as this is a general solving scheme for combinatorial optimization problems. One of the advantage of staying general is the accessibility to solvers. Solvers implement readily available state-of-the-art algorithms while staying highly customizable to a specific problem and/or individual instances. As research evolves, new algorithms are discovered and implemented in existing solvers leading to an improved performance on novel and known models alike. On the other hand, it remains that solvers are complex machinery that may require some problem-specific fine tuning on the hardest problems or instances.

Nonetheless, even without any fine tuning, these highly efficient all-purpose algorithmic machines come with “optimality guarantee”.[1] As such, part of the price of using solvers is the price of optimality.

We chose CP among other modeling techniques because of its expressiveness. CP has a tremendous amount of constraints that allows for models that naturally fit the definition of a problem. CP solvers are also known to be highly customizable, a property we propose to exploit, in this chapter, by using our novel TD heuristic for the OSP as a value selection heuristic over the searcher’s path variables. Furthermore, CP turned out to be effective in solving search theory problems (e.g., the multiple rectangular search areas problem [Abi-Zeid et al., 2011b]). Finally, CP allows to find high quality solutions quickly, an interesting property on the hardest problems and/or instances where the price of optimality is high.

The OSP problem is a path planning problem with uncertain goal (in our case search object) location. It is a path planning problem under uncertainty. According to [Brown and Miguel, 2006], uncertainty in constraint problems may arise in two situations:

- the problem changes over time (*dynamically changing* problems), and
- some problem’s data or information are missing or are unclear (*uncertain* problems).

The OSP problem formulation as a constraint program is not uncertain in this sense since it has a complete description. Nonetheless, the location of the search object, its detectability, and its motion are represented by probability distributions. Our CP is not a dynamic formulation since the searcher’s detection model, the object’s motion model, and the prior probability distribution on its location are known a priori. More specifically, the OSP problem is a path planning problem with a Markov Decision Process formulation that uses negative information for updating the probabilities in the absence of detection. In the case where the total number of plausible search object’s paths is sufficiently low, a situation that rarely occurs in realistic search problems, the problem could potentially be formulated using multiple scenarios and thus be considered a stochastic CP (e.g., [Tarim et al., 2006]) where a scenario would correspond to a possible path of the search object. However, this is not an interesting approach since the Markov OSP problem specialization from search theory enables us to solve the problem without enumerating all the object’s plausible paths [Brown, 1980]. Surveys on dealing with uncertainty in constraint problems may be found in [Brown and Miguel, 2006, Verfaillie and Jussien, 2005].

[1] Most solvers in mathematical and constraint programming come with “optimality guarantee” if enough resources are available. There exists, however, a growing population of different types of solvers based on local searches and metaheuristics [Hoos and Tsang, 2006]. These solvers, although highly efficient, do not always guarantee the optimality of a solution.

We describe, in the first part of this chapter, our novel CP model for the OSP problem (Section 4.1). Subsection 4.1.2 presents experiments using two different objective functions. The TD heuristic is described in Section 4.2. The experimental results of Subsection 4.2.2 confirm that the solver achieves a superior performance when using the TD heuristic than with CP along with general purpose heuristics. We conclude and give further research avenues in Section 4.3.

4.1 Constraint Programming for Search Path Planning

We present, in this section, the CP model we developed for the OSP. The model uses the following constants from the definition of the problem (see Sections 2.2.2 and 2.2.3):

- T, the number of available time steps and $\{1,\ldots,T\}$ the set of all time steps;
- $G_A = (\mathcal{V}(G_A), \mathcal{E}(G_A))$, the accessibility graph that represents the search environment;
- $y_0 \in \mathcal{V}(G_A)$, the initial position of the searcher;
- $poc_1(r)$, the initial probability of containment in vertex r ($\forall r \in \mathcal{V}(G_A)$);
- $pod(r)$, the conditional probability of detecting the object when the position y_t of the searcher is r ($\forall t \in \{1,\ldots,T\}, \forall r \in \mathcal{V}(G_A)$);
- $\mathbf{M}_{sr}$, the probability of an object's move from vertex s to vertex r in one time step ($\forall s, r \in \mathcal{V}(G_A)$).

The Variables. The decision variables are the searcher's position at each time step used to define the search plan P:

- $PATH_t \in \mathcal{V}(G_A)$, the searcher's position at time t ($\forall t \in \{1,\ldots,T\}$) with $PATH_0 = y_0$.

The domain of $PATH_t$ is the set of vertices of the graph and is therefore finite ($\forall t \in \{1,\ldots,T\}$). The probability variables are the following:

- $POC_1(r) = poc_1(r)$, the probability of containment in vertex r at time 1 ($\forall r \in \mathcal{V}(G_A)$);
- $POC_t(r) \in [0,1]$, the probability of containment in vertex r at time t ($\forall t \in \{1,\ldots,T\}, r \in \mathcal{V}(G_A)$);
- $POS_t(r) \in [0,1]$, the probability of success in vertex r at time t ($\forall t \in \{1,\ldots,T\}, r \in \mathcal{V}(G_A)$);
- $COS \in [0,1]$, the *cos* criterion value, i.e., the sum of all local probabilities of success up to time T.

The domains of the probability variables are infinite since these variables are real. Interval-valued domains are used to define these domains, i.e., non-enumerated domains whose values are implicitly given by a lower bound and an upper bound.

The Constraints. Constraint (4.1) constrains the searcher to move from one vertex to another according to the accessibility graph edges $\mathcal{E}(G_A)$.

$$(PATH_{t-1}, PATH_t) \in \mathcal{E}(G_A), \qquad \forall t \in \{1, \ldots, T\}. \tag{4.1}$$

Constraints (4.2) to (4.4) compute the probabilities required to evaluate the *cos* criterion. The first two constraints, (4.2) and (4.3), compute the probabilities of success. Constraint (4.4) is the containment update equation.

$$PATH_t = r \implies POS_t(r) = POC_t(r)pod(r), \qquad \forall t \in \{1, \ldots, T\}, \forall r \in \mathcal{V}(G_A). \tag{4.2}$$

$$PATH_t \neq r \implies POS_t(r) = 0, \qquad \forall t \in \{1, \ldots, T\}, \forall r \in \mathcal{V}(G_A). \tag{4.3}$$

$$POC_t(r) = \sum_{s \in \mathcal{V}(G_A)} \mathbf{M}_{sr}\left[POC_{t-1}(s) - POS_{t-1}(s)\right], \qquad \forall t \in \{2, \ldots, T\}, \forall r \in \mathcal{V}(G_A). \tag{4.4}$$

We have experimented with two different encodings of the objective function. The first one encodes the objective function as a *sum*, the second one encodes it as a *max*. Both encodings are equivalent and lead to the same objective value.

The *sum* Objective Function. The sum encoding of Equation (4.5) consists of encoding the objective function as it appears in Equation (2.7). It is the natural way to represent this function.

$$\begin{aligned} &\max COS, \\ &COS = \sum_{t \in \{1,\ldots,T\}} \sum_{r \in \mathcal{V}(G_A)} POS_t(r). \end{aligned} \tag{4.5}$$

The *max* Objective Function. The sum constraint does a very poor job of filtering the variables: the upper bound on a sum of variables is given by the sum of the upper bounds of the domains of the variables. However, since we know that in the summation $\sum_r POS_t(r)$ only one variable is non-null, a tighter upper bound on this sum is given by the maximum domain upper bound. A tighter upper bound on the objective variable generally leads to a faster B&B in the solver. We therefore have implemented the objective function defined by Equation (2.7) using the following constraints:

$$\begin{aligned} &\max COS, \\ &COS = \sum_{t \in \{1,\ldots,T\}} \max_{r \in \mathcal{V}(G_A)} POS_t(r). \end{aligned} \tag{4.6}$$

Example 4.1.1 (Objective function filtering)**.** Suppose the number of time steps is $T = 2$ and that the number of vertices is $|\mathcal{V}(G_A)| = 2$. Further assume the following domains for the probability of success variables:

$$\mathrm{dom}(POS_1(1)) = [.0, .2], \tag{4.7}$$

$$\mathrm{dom}(POS_1(2)) = [.3, .4], \tag{4.8}$$

$$\mathrm{dom}(POS_2(1)) = [.0, .1], \tag{4.9}$$

$$\mathrm{dom}(POS_2(2)) = [.1, .2]. \tag{4.10}$$

One of the solver's tasks is to perform a filtering of the objective function upper bounds using its related constraints. Given the constraints of Equation (4.5), the upper bound on the objective is the double sum of the upper bounds of the success probabilities:

$$\begin{aligned}\overline{COS} &= \sum_{t\in\{1,\ldots,T\}} \sum_{r\in\mathcal{V}(G_A)} \overline{POS}_t(r)\\ &= 0.2 + 0.4 + 0.1 + 0.2 = 0.9.\end{aligned} \tag{4.11}$$

This leads to an upper bound of 0.9. Given the constraints of Equation (4.6), the upper bound on the objective is the sum, over all time steps, of the maximal value of the upper bounds of the success probabilities across vertices:

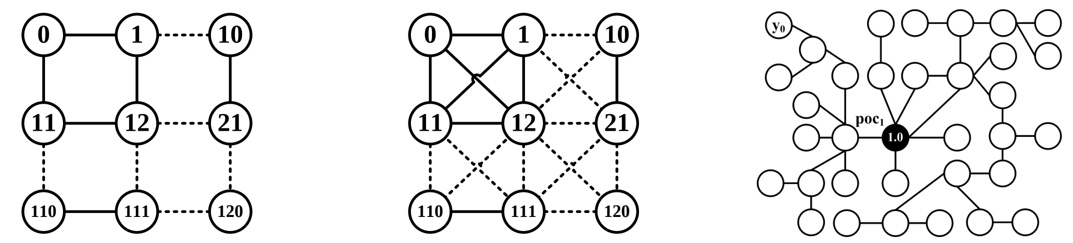
$$\begin{aligned}\overline{COS} &= \sum_{t\in\{1,\ldots,T\}} \max_{r\in\mathcal{V}(G_A)} \overline{POS}_t(r)\\ &= \max(0.2, 0.4) + \max(0.1, 0.2)\\ &= 0.4 + 0.2 = 0.6.\end{aligned} \tag{4.12}$$

This leads to an upper bound of 0.6 which is lower than the previous value of 0.9 thus leading to a tighter upper bound on the objective. ◁

Domain pre-filtering. The updated probability of containment in vertex r at time t in the absence of searches, $poc_t^{\text{Markov}}(r)$, is:

$$poc_t^{\text{Markov}}(r) \overset{\text{def}}{=} \begin{cases} poc_1(r), & \text{if } t = 1;\\ \sum_{s\in\mathcal{V}(G_A)} \mathbf{M}_{sr}\, poc_{t-1}^{\text{Markov}}(s), & \text{otherwise.}\end{cases} \tag{4.13}$$

The Markovian probability of containment poc^{Markov} is an upper bound on the probability of containment in vertex r at time t, i.e., $poc_t(r) \leq poc_t^{\text{Markov}}(r)$. This is due to the fact that an unsuccessful search in vertex r at time t decreases the probability of the object being there at time t (from Equation (2.11)). Moreover, we observe that the probability of success $pos_t(r)$ in vertex r at time t is bounded by the probability of detection in r ($pod(r)$), i.e., $pos_t(r) \leq pod(r)$. Both of these observations will be used to filter the domains of the probability variables in the CP model prior to the solving process. This leads to the following pre-filtered domains:



(a) The 11 × 11 grid G^+ (b) The 11 × 11 grid G^* (c) The graph G^L

Figure 4.1: The search environments

- $POC_t(r) \in [0, poc_t^{\text{Markov}}(r)]$, the probability of containment in vertex r at time t ($\forall t \in \{1, \ldots, T\}, r \in \mathcal{V}(G_A)$) where $poc_t^{\text{Markov}}(r)$ is defined by Equation (4.13);
- $POS_t(r) \in [0, pod(r)]$, the probability of success in vertex r at time t ($\forall t \in \{1, \ldots, T\}, r \in \mathcal{V}(G_A)$);

4.1.1 Experiments on the Objective Function

In the next experiments, we compare the objective functions defined in Section 4.1. We call the model that implements the double sum objective function of Equation (4.5) *OSP-SUM* and the model that implements the novel objective function of Equation (4.6) *OSP-MAX*. We compare, in the first part of the experiments, the results obtained with both objective functions. We use the following static ordering of the decision variables for branching: $PATH_1, PATH_2, \ldots, PATH_T$. This is the natural order for branching in an OSP. Then, the solver is configured to apply a simple decreasing domain value selection heuristic. In the second part of the experiments, we present additional results obtained when using the impact-based search branching heuristic (strategy) instead of a static ordering of the path variables. The impact-based search branching heuristic [Refalo, 2004] is a state-of-the-art general-purpose branching heuristic for CP. We compare the performance of the *OSP-SUM* model to the performance of the *OSP-MAX* model when both are used along with impact-based search. The models that use impact-based search are suffixed by *IB*, i.e., *OSP-SUM-IB* for the model of Equation (4.5) and *OSP-MAX-IB* for the model of Equation (4.6).

The graphs (or search environments) used in our benchmark are shown on Figure 4.1. G^+ is a reflexive 11 × 11 grid where all adjacent vertices except diagonals are linked by an edge (see Figure 4.1(a)). G^* is a reflexive 11 × 11 grid where all adjacent vertices (diagonals included) are linked by an edge (see Figure 4.1(b)). In both grids, the searcher's initial position y_0 is vertex 0 and the initial probability of containment distribution is such that $poc_1(60) = 1$, i.e., the object is in the middle of the grid. G^L is a reflexive graph generated using the Université Laval tunnels map. It is almost a tree. The only cycle is in the middle of the graph, near the vertex with a non-null probability of containment. The searcher starts in the upper left corner of the graph. The object starts roughly in the middle of the graph (see Figure 4.1(c)).

The assumed Markovian object's motion model is

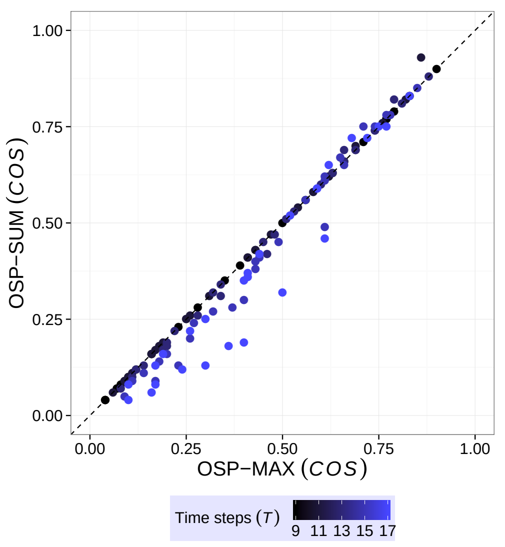
$$\mathbf{M}_{sr} = \begin{cases} \frac{1-\rho}{\deg(s)-1}, & \text{if } (s,r) \in \mathcal{E}(G_A), \\ \rho, & \text{if } s = r, \end{cases} \tag{4.14}$$

where $\deg(s)$ is the degree of vertex s and $\rho \in \{0.3, 0.6, 0.9\}$ is a parameter of the instance indicating the probability that the object stays in its current location. We tried these graphs with three different probabilities of detection: $pod(r) \in \{0.3, 0.6, 0.9\}$ $(\forall r \in \mathcal{V}(G_A))$. The total times allowed for the searches are $T \in \{9, 11, 13, 15, 17\}$. Usual OSP problem experiments use grids similar to G^+ (e.g., [Eagle and Yee, 1990], [Lau et al., 2008]). Therefore, our G^+ problem instances are comparable with those used in the literature. The decision space of a G^* instance is larger than the one of a G^+ instance since the degree of each node is higher. The G^L instance is closer to a building-like environment.

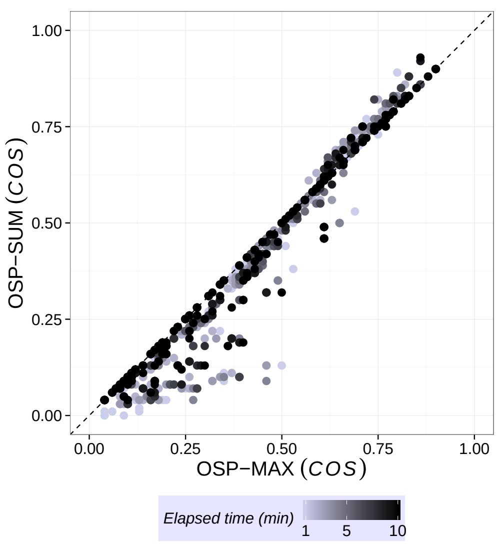
All implementations were done using Choco Solver 2.1.5 [Laburthe and Jussien, 2012], a solver developed in the Java programming language, and the Java Universal Network/Graph (JUNG) 2.0.1 framework [O'Madadhain et al., 2010]. The probabilities of the OSP CP model were mapped from reals in $[0, 1]$ to integers in $[0, U]$. We chose $U = 10^4$ since it allowed for enough precision while avoiding integer overflows in Choco Solver. All tests consisted of a single run on an instance $\left(G_A, T, pod(r)_{r\in\mathcal{V}(G_A)}, \rho\right)$, as described above. Computations were made on the supercomputer Colosse from Université Laval. We allowed a total number of 5,000,000 backtracks and a time limit of 10 minutes. We disabled restarts for all runs except when using a model along with the impact-based search branching heuristic, i.e., for the *OSP-SUM-IB* model and the *OSP-MAX-IB* model. Restarts are known to improve the performance of impact-based search [Refalo, 2004].

4.1.2 Results and Discussion

Figure 4.2 is a comparison of the objective values obtained by the solver when using the *OSP-SUM* model against the objective values obtained by the solver when using the *OSP-MAX* model. The figure is composed of two subfigures. Each dot on the first subfigure (see Figure 4.2(a)) is the comparison of the objective value of the two incumbent solutions found by both methods on a single OSP problem instance within 10 minutes (or 5,000,000 backtracks). A dot lying on the dashed frontier represents an instance for which the solver achieved the same objective value with both models. A dot on the right-hand side (resp. left-hand side) of the frontier represents an instance for which the *OSP-MAX* model (resp. *OSP-SUM* model) outperformed the *OSP-SUM* model (resp. *OSP-MAX* model). We represent the complexity of the instance, in terms of allowed time steps T, by a gradient from black to light blue. Light blue dots belong to problem instances with $T = 17$ whereas black dots belong to problem instances with $T = 9$. The subfigure on the right-hand side (see Figure 4.2(b)) enables us to see the evolution of the objective values of the best so far incumbent solutions as the



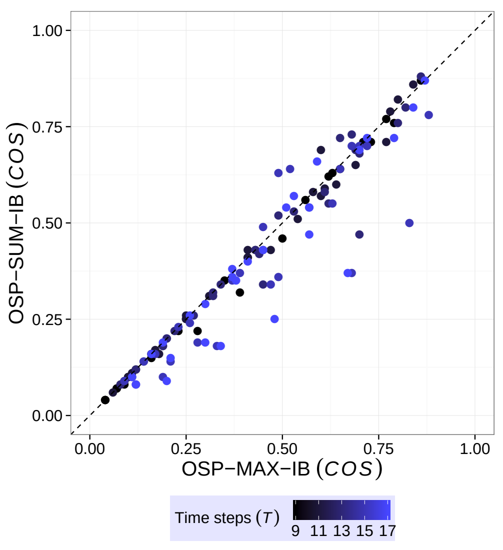
(a) *OSP-SUM* against *OSP-MAX*: Last incumbent objective value within 10 minutes; the complexity of the instance (in terms of allowed time steps T) is represented by a gradient from black to blue.



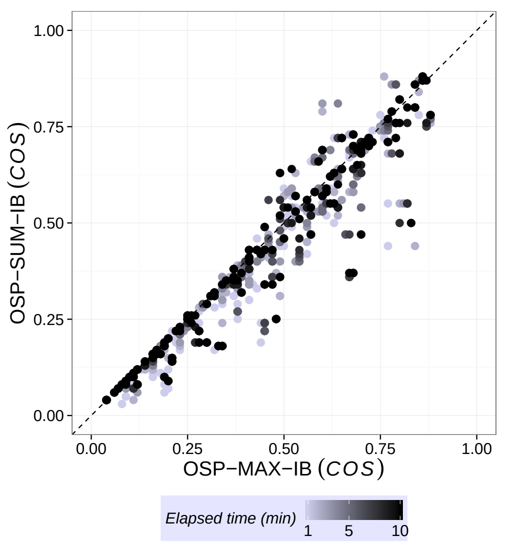
(b) *OSP-SUM* against *OSP-MAX*: Last incumbent objective value in time; the higher the allowed time is, the darker the dot is.

Figure 4.2: Comparison of the objective values obtained by the solver when using the *OSP-SUM* model to the objective values obtained by the solver when using the *OSP-MAX* model; each dot is a comparison of the incumbent solutions found by the compared methods on a single OSP problem instance.

solver runs. We took a total of 10 snapshots at intervals of one minute during the solving process. At each snapshot, we recorded the objective value of the best so far incumbent solution. The second snapshot, for instance, records the objective value of the best so far incumbent solution with an allowed solving time of two minutes. The darker a dot is, the larger the allowed solving time is. We first notice the general tendency of the *OSP-MAX* model to outperform the *OSP-SUM* model on most instances. This tendency is clear at the beginning of the solving process (see the dots with light shading on Figure 4.2(b)). The performance of the solver is similar with both models on simpler instances, i.e., when T is low (Figure 4.2(a)). It is, in the majority of the cases, in favor of *OSP-MAX* for larger values of T. This supports the fact that the *OSP-MAX* model objective function definition is an improvement over the usual OSP objective function definition used in the *OSP-SUM* model. Figure 4.3 is a comparison similar to Figure 4.2, but in presence of the impact-based search branching heuristic. Figure 4.3 shows a tendency that is similar to the one we observed on Figure 4.2. That is, for more complex instances and early during the solving process, the *OSP-MAX-IB* model's objective function definition tends to improve the performance of the solver with respect to that of the *OSP-SUM-IB* model.



(a) *OSP-SUM-IB* against *OSP-MAX-IB*: Last incumbent objective value within 10 minutes; the complexity of the instance (in terms of allowed time steps T) is represented by a gradient from black to blue.



(b) *OSP-SUM-IB* against *OSP-MAX-IB*: Last incumbent objective value in time; the higher the allowed time is, the darker the dot is.

Figure 4.3: Comparison of the objective values obtained by the solver when using the *OSP-SUM-IB* model to the objective values obtained by the solver when using the *OSP-MAX-IB* model; each dot is a comparison of the incumbent solutions found by the compared methods on a single OSP problem instance.

4.2 The Total Detection Heuristic as a Value Selector

In this section we describe the TD heuristic. The idea of the TD heuristic is based on a stochastic generalization of a graph theory pursuit-evasion problem called the *cop and robber* game [Nowakowski and Winkler, 1983]. We were also inspired by a domain ordering idea that was successfully used for the *multiple rectangular search areas problem*, a search theory problem studied in [Abi-Zeid et al., 2011b]. We first published the heuristic in [Morin et al., 2012]. Since then, it has been extended to produce a bound for the OSP [Simard et al., 2014, 2015]. We present it as it was first published in [Morin et al., 2012]. That is, as a value selection heuristic for the searcher's path variables.

The general idea is to simplify the probability system in the OSP problem by ignoring the negative information received by the searcher when s/he fails to detect the object. That is, at each time step $t \in \{1, \ldots, T\}$, the heuristic chooses the most promising accessible vertex based on the total probability of detecting the object in the remaining time.

Let $G_A = \langle \mathcal{V}(G_A), \mathcal{E}(G_A) \rangle$ be the accessibility graph where the searcher and the object evolve. Let $t \in \{1, \ldots, T\}$ be a time step, $y \in \mathcal{V}(G_A)$ be the position of the searcher, and and $o \in \mathcal{V}(G_A)$ be the position of the object. Let $w_t(y, o)$ be the conditional probability that the searcher detects the object in the time period $[t, t+1, \ldots, T]$ given that, at time t, the searcher is in y and the object in o. The function $w_t(y, o)$, which is backward recursive, is defined as follows:

$$w_t(y,o) \overset{\text{def}}{=} \begin{cases} pod(o), & \text{if } o = y \text{ and } t = T, \\ 0, & \text{if } o \neq y \text{ and } t = T, \\ pod(o) + (1 - pod(o))p_t(y,o), & \text{if } o = y \text{ and } t < T, \\ p_t(y,o), & \text{if } o \neq y \text{ and } t < T, \end{cases} \tag{4.15}$$

where

$$p_t(y,o) = \sum_{o' \in \mathcal{N}(o)} \mathbf{M}_{oo'} \max_{y' \in \mathcal{N}(y)} w_{t+1}(y', o') \tag{4.16}$$

is the probability of detecting the object in the period $[t+1, \ldots, T]$. Equations (4.15) and (4.16) have the following interpretation:

- If $t = T$, the searcher has a probability $pod(o)$ of detecting the object when the searcher and the object are co-located, i.e., $o = y$; otherwise, the searcher and the object are not co-located and the probability is null.

- If $t < T$ and $o = y$, then the searcher can detect the object at time t with probability $pod(o)$ or fail to detect it at time t with probability $1 - pod(o)$. If the searcher fails to detect the object at time t, s/he may detect it during the period $[t+1, \ldots, T]$. The probability of detecting the object in the period $[t+1, \ldots, T]$ is given by $p_t(y, o)$ (Equation (4.16)) and may be interpreted as follows:

  – In the case where there is only one edge leaving vertex o to vertex o', the searcher chooses the accessible vertex y' that maximizes the conditional probability of detecting the object in the time period $[t+1, \ldots, T]$, given her/his new position y' and the new object's position o', i.e., $\max_{y' \in \mathcal{N}(y)} w_{t+1}(y', o')$.

  – In the general case where vertex o has many neighbors, $p_t(y, o)$ is the average of all the maximal $w_{t+1}(y', o')$ weigthed by the probability $\mathbf{M}_{oo'}$ of moving from o to o'.

  This is reasonable since we do not control the object's movements but we can move the searcher to the vertex that has the highest probability of success.

- Finally, if the search time is not over (i.e., $t < T$) and the object and the searcher are not co-located (i.e., $o \neq y$), the probability of detecting the object at time t is null and the probability of success depends entirely on the probability $p_t(y, o)$ of detecting the object within the period $[t+1, \ldots, T]$.

A searching strategy $S : \{1, \ldots, T\} \times \mathcal{V}(G_A) \to \mathcal{V}(G_A)$ assigns to each time step and searcher's position a vertex that is considered to be promising according to some heuristic. In the TD heuristic case, the strategy sets the new searcher's position to be the accessible vertex that maximizes the probability of detecting the object in the remaining time:

$$S_t(PATH_t) \overset{\text{def}}{=} \underset{y' \in \text{dom}\,(PATH_t)}{\text{argmax}} \sum_{o \in \mathcal{V}(G_A)} w_t(y', o) POC_t(o), \qquad \forall t \in \{1, \ldots, T\}. \tag{4.17}$$

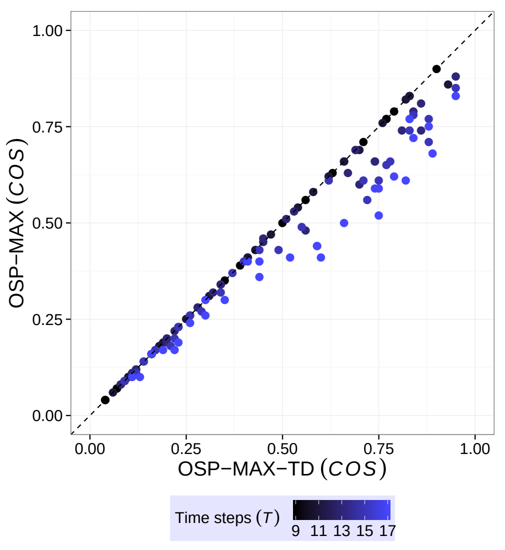
In order to apply this value selection heuristic, the following *static ordering* of the decision variables is used: $PATH_1, PATH_2, \ldots, PATH_T$. That is, the solver branches first on $PATH_1$, then on $PATH_2$ and so on. Each time the solver branches on a new variable $PATH_t$, the strategy $S_t(PATH_t)$ is computed in polynomial time. To do so, we solve the recurrence relation of Equation (4.15) during the generation of the model. We keep in memory a total of T square matrices to store the TN^2 required probabilities. Let $\mathbf{W}^{(t)}$ be such a matrix. Solving Equation (4.17) is now straightforward. Since the $\mathbf{W}^{(t)}$ matrices are pre-computed, the complexity of Equation (4.17) no longer depends on the time steps T.

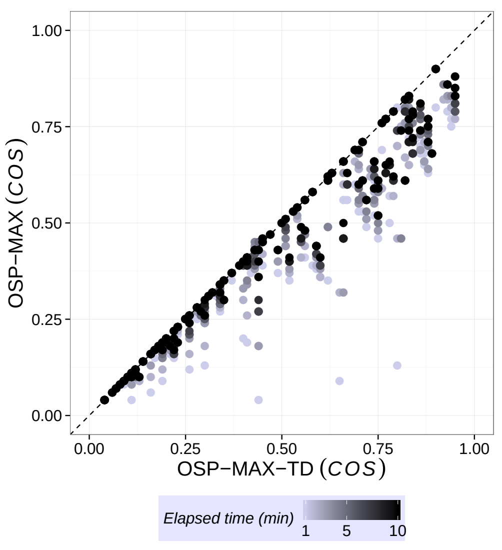
4.2.1 Experiments on the Total Detection Heuristic

We showed, in section 4.1.1, that an objective function using Equation (4.6) triggers more filtering than an objective function based on Equation (4.5). We thus retained the *OSP-MAX* model from the experiments of Section 4.1.1. We show, in the next experiments, the benefits of using the TD heuristic as a value selection heuristic in a CP model of the OSP problem. To do so, we added the TD heuristic on top of the *OSP-MAX* model to select the next destination of the searcher when branching. We call this model *OSP-MAX-TD*. We first compare the performance of the *OSP-MAX-TD* model to the one of the *OSP-MAX* model which does not use any problem-specific value selection heuristic but a simple decreasing domain general heuristic. Both models branch on the path variables in their natural order, i.e., in the order of the time steps. Second, we compare the performance of the *OSP-MAX-TD* model to the one of the *OSP-MAX-IB* model which uses the impact-based search heuristic [Refalo, 2004]. The experimental framework is the same as the one presented in Section 4.1.1. All tests consisted of a single run on an instance $\left(G_A, T, pod(r)_{r \in \mathcal{V}(G_A)}, \rho\right)$, as defined in Section 4.1.1. We allowed a total number of 5,000,000 backtracks and a time limit of 10 minutes. Computations were made on the supercomputer Colosse from Université Laval.

4.2.2 Results and Discussion

We compare, on Figure 4.4, the objective values of the incumbent solutions found by the solver with and without the TD heuristic. Figure 4.4(a) is a two-by-two comparison of each incumbent solution found within 10 minutes. Each dot compares the objective value achieved by the solver when using the two compared methods on a single problem instance. Dots that fall on the right-hand side of the dashed frontier belong to problem instances for which

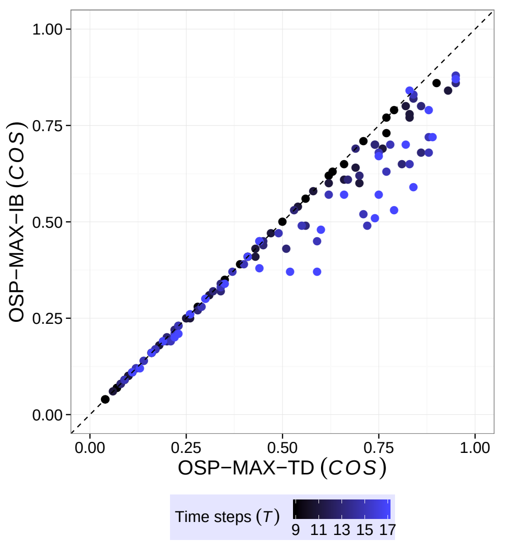


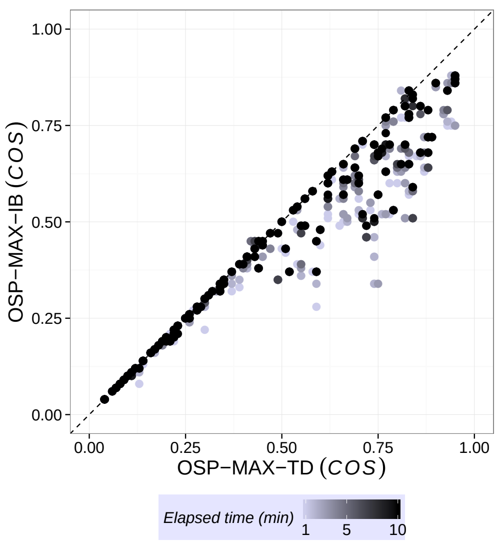


(a) *OSP-MAX* against *OSP-MAX-TD*: Last incumbent objective value within 10 minutes; the complexity of the instance (in terms of allowed time steps T) is represented by a gradient from black to blue.

(b) *OSP-MAX* against *OSP-MAX-TD*: Last incumbent objective value in time; the higher the allowed time is, the darker the dot is.

Figure 4.4: Comparison of the objective values obtained by the solver when using the *OSP-MAX* model to the objective values obtained by the solver when using the *OSP-MAX-TD* model; each dot is a comparison of the incumbent solutions found by the compared methods on a single OSP problem instance.





(a) *OSP-MAX-IB* against *OSP-MAX-TD*: Last incumbent objective value within 10 minutes; the complexity of the instance (in terms of allowed time steps T) is represented by a gradient from black to blue.

(b) *OSP-MAX-IB* against *OSP-MAX-TD*: Last incumbent objective value in time; the higher the allowed time is, the darker the dot is.

Figure 4.5: Comparison of the objective values obtained by the solver when using the *OSP-MAX-IB* model to the objective values obtained by the solver when using the *OSP-MAX-TD* model; each dot is a comparison of the incumbent solutions found by the compared methods on a single OSP problem instance.

the TD heuristic improved the solver's performance. Dots that lie on the frontier are ties considering the allowed solving time. The bluer a dot representing an OSP problem instance is, the larger the total number of allowed time steps for that problem instance is, i.e., the black dots stand for the lowest value of T whereas the blue dots stand for larger values of T. Figure 4.4(b) presents the same information at various points in time. The darker a dot is on this figure, the larger the allowed solving time is. Similar figures are presented in Section 4.1.2 to assess the performance of the solver with respect to the choice of objective function. The figures clearly show how much using the TD heuristic to select the next destination of the searcher helps in finding high quality incumbent solutions quickly on the various instances of our benchmark. There are, on Figure 4.4(b), many light gray dots on the right-hand side of the dashed frontier meaning that the solver has found an incumbent of high quality near the beginning of the solving process when using TD while failing to do so without the heuristic. The benefits of the TD heuristics become clear for T large (see Figure 4.4(a)). Similar results are obtained when comparing the objective value of the incumbent solutions obtained when using impact-based search, i.e., with the *OSP-MAX-IB* model, to the one obtained when using the TD heuristic as a value selection heuristic, i.e., with the *OSP-MAX-TD* model (see Figure 4.5). We allowed the solver to use restarts when using impact-based search as it is known to improve the performance of the heuristic [Refalo, 2004]. We disabled restarts with the TD heuristic. Again, *OSP-MAX-TD* is a clear winner.

4.3 Further Discussion on the Total Detection Heuristic

In this chapter, we used the TD heuristic to select the next destination of the searcher in a CP model. Even though we centered the discussion on CP, we found out, in additional experiments, that the TD heuristic is efficient when used in conjunction with other solving techniques as well. We could use it, for instance, in mixed-integer linear programming (MILP). For this purpose, we would need to implement a MILP model similar to the one developed in [Morin, 2010] for the OSP problem with a visibility criterion which is a generalization of the OSP problem [Morin et al., 2009]. Using TD in a MILP could be done, for instance, by providing the heuristic's solution as a starting point for the MILP solver. It is also possible to use TD as a branching heuristic directly in the solver. This second approach is similar to what we have done in CP. Both methods showed positive results indicating that adapting the TD heuristic to other solving techniques is not only feasible but also valuable. This is, however, left as an open avenue for further research since we focus, in this chapter, on CP.

That being said, the benefits of using the TD heuristic as a value selection heuristic for the OSP problem are clear. The solver finds incumbent solutions of high quality in short time when using the TD heuristic as a value selection heuristic for the path variables. In order to get an idea of how the TD heuristic behaves in our CP model, we can compare our results to the ones published in the literature using an OSP specific B&B algorithm [Lau et al., 2008].

Table 4.1: The time to last incumbent on a 11×11 G^+ grid with $pod(r) = 0.6$ and $\rho = 0.6$ compared to the time spent by a B&B procedure to prove optimality when using various bounds [Lau et al., 2008].

T	*Time to optimality (s)*				
	Constraint Programming	B&B algorithm with various bounds†			
	OSP-MAX-TD	DMEAN	MEAN	PROP	FABC
15	4	3	12	8	63
17	39	24	72	37	353

† The time values for the B&B algorithm are taken from [Lau et al., 2008].

After communicating with the authors of [Lau et al., 2008] we were able to validate that our solutions for the G^+ instances are optimal up to the fourth decimal notwithstanding the mapping of the domains of the probability variables to integer bounded domains required for implementation purposes on Choco 2.1.5 (see Section 4.1.1). Table 4.1 presents the time to last incumbent on a 11×11 G^+ grid with $pod(r) = 0.6$ and $\rho = 0.6$, and the time spent by a B&B procedure to prove the optimality of its last incumbent solution when using various bounds from the literature. We provided, in this chapter, a simple improvement of the definition of the objective function to help the solver to prove the optimality of its incumbent solution (see Equation (4.6)). The solver, however, did not prove the optimality of its last incumbent solution for the largest values of T. A novel bound for the OSP problem is provided as part of further research [Simard et al., 2014, 2015]. Such a bound greatly helps in pruning branches of the B&B tree of the CP solver.

Finally, even though we used the TD heuristic to solve the OSP problem, it mainly deals with two general notions. The first being the notion of transition probabilities defined between the various states of a given state space. The second being the notion of a sequence of decisions which influences the transition probabilities evolution in time and possibly a given objective function. These decisions, in the OSP, are the constrained positions of the searcher, but the concepts behind TD are general. Further challenges thus include the application of the heuristic to other problems.

Chapter 5

Markov Transition Constraint

This chapter is based on our original work published in [Morin and Quimper, 2014] and presented at the 11th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming (CP-AI-OR 2014).

We present, in this chapter, a novel global *Markov transition constraint* (Mtc) to model finite state space Markov processes with a finite number of steps T. Such processes arise in many contexts including the modeling of the motion of a search object (see Section 1.4 of Chapter 1 for an introduction to Markov chains). This chapter thus builds on the contributions of the previous one while providing a general contribution to CP. In a first study, we discuss cases where elementary arithmetic constraints enforce bounds consistency (see Definition 1.2.19 from Section 1.2.1 of Chapter 1). We show, both theoretically and empirically, that a set of elementary arithmetic constraints decomposing an Mtc is insufficient to enforce bounds consistency in all cases. This justifies the need for a global filtering of Markov chains and thus the introduction of the Mtc for both performance and modeling purposes. Apart from a discussion on the capabilities of interval arithmetic filtering with and without implied constraints[1], we introduce two global filtering algorithms for this constraint. The first one is based on linear optimization. The second one is inspired from a fractional knapsack algorithm. Linear programming always performs bounds consistency. It is, however, an expensive approach since a constraint solver needs to run the filtering algorithms on an exponential number of nodes. We thus developed a filtering algorithm based on the fractional knapsack problem. The method is proved to achieve bounds consistency when the transition matrix is monomial and in the case of forward reasoning[2]. The class of filtering problems with forward

[1] Implied constraints are redundant constraints. They fit the definition of the problem, but they are not necessary. Adding such constraints to a CP model is known to improve filtering [Smith, 2006].

[2] The expressions forward reasoning and backward reasoning as we use them in this chapter are not to be confused with the forward and backward equations of a Markov chain [Cox and Miller, 1972]. In both forward (backward) reasoning and the forward (backward) equation we compute information at step t using information from step $t' < t$ ($t' > t$). However, in both forward and backward reasoning we aim at computing a probability distribution over the states whereas the backward equation deals with a notion of expected rewards.

reasoning arises in many applications. In a second study and to conclude, we apply the Mtc global constraint on a practical case of path planning under uncertainty: the OSP problem.

Example 5.0.1. We computed, in Example 1.4.1 (see Section 1.4 of Chapter 1), the probability distribution on a lost child's location in a search environment made of three states: a toys store (state 1), a candy store (state 2), and a food market (state 3). The child's motion, discretized on a 1 minute time scale, has been modeled as the following transition matrix:

$$\mathbf{M} = \begin{bmatrix} \frac{7}{8} & \frac{1}{8} & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 1 & 0 \end{bmatrix}. \tag{5.1}$$

Suppose that we have no information on the child's initial location, i.e., the probability distribution at time 1 represented by vector $\mathbf{x}^1$ is uncertain or unknown. We might want to use a uniform distribution to model the location of the child. This would be assuming that the location probability of the child is uniform across the states at time 1. We want to avoid such an assumption as it does not reflect our reality. Instead, we want to obtain the best approximation on the location of the child after 1 minute, i.e., at time $t = 2$. The only thing we know is that $\mathbf{x}^1$ is a distribution: it sums up to 1 and the probabilities are in the interval $[0, 1]$. That is,

$$\mathbf{x}^1 = [[0,1],[0,1],[0,1]]\,. \tag{5.2}$$

The problem we want to solve is still to compute $\mathbf{x}^2$ by using the scalar product between $\mathbf{x}^1$ and $\mathbf{M}$:

$$\mathbf{x}^2 = \mathbf{x}^1\mathbf{M}. \tag{5.3}$$

However, both $\mathbf{x}^1$ and $\mathbf{x}^2$ are vectors of interval probabilities. One simple solution is to rely on interval arithmetic. By simply multiplying $\mathbf{x}^1$ by $\mathbf{M}$ we find that the probabilities of locating the child in store 1, 2 or 3 are in the intervals: $[0, 1]$, $[0, 1]$, and $\left[0, \frac{1}{3}\right]$ respectively. These are valid bounds, but these are not the tightest possible intervals. By relying on linear optimization techniques or on the fractional knapsack filtering algorithm we present in Section 5.2, we find that, the probabilities of locating the child in store 1, 2 or 3 are in the intervals $\left[0, \frac{7}{8}\right]$, $\left[\frac{1}{8}, 1\right]$, and $\left[0, \frac{1}{3}\right]$ respectively. These are the tightest possible bounds which are our best possible approximation on the location of the child. This is a solution that interval arithmetic alone is not able to provide. Intuitively, we see, by inspection of the transition matrix $\mathbf{M}$, that the child moves to store 3 within a minute with a probability of at most $\frac{1}{3}$ since there is only one non-zero probability of transition to store 3. Thus, $\frac{1}{3}$ is the maximal probability that we locate the child in store 3 after a minute, i.e., at $t = 2$. Furthermore, the child cannot be located in store 1 with a probability greater than $\frac{7}{8}$. The only ways to reach store 1 is either by moving from store 2 or by staying there if we are already there. The former event has a probability of $\frac{1}{3}$ while the latter has a probability of $\frac{1}{8}$. The probability distribution on the location of the child at time $t = 1$, i.e. $\mathbf{x}^1$, sums to 1 by definition. Thus, the probability that we locate the child in store 3 at time $t = 2$ is no more than $\frac{7}{8}$ which is achieved when the child is in store 1 with certainty at time $t = 1$ and stays there. Finally, at $t = 2$ the child is in store 2 with a probability of at least $\frac{1}{8}$ since $\mathbf{x}^2$ sums up to 1 and since s/he is elsewhere with a probability of at most $\frac{7}{8}$. $\triangleleft$

Clearly, filtering uncertain probability distributions (Definition 1.4.5) to the tightest possible intervals (Example 5.0.1) is not as easy as computing the probabilities with a known probability distribution (Example 1.4.1). Uncertainty on probability distributions arises in modeling and problem solving due to external factors influencing the chain, imprecise data, and/or uncertain knowledge. For instance, in the search operation model presented in Section 5.3, the probability distributions are updated given the searcher's actions. There is no way, without knowing the entire searcher's path, to compute the exact probability distribution for each time step. Nonetheless, a CP solver needs to compute the tightest possible intervals for the probabilities.

The rest of the chapter is organized as follows. We define the Mtc global constraint in Section 5.1. Filtering methods for the Mtc are presented in Section 5.2. We present, in that section, both theoretical and empirical results for the problem of filtering a single Mtc. An application of the Mtc constraint to the OSP problem is presented in Section 5.3. We conclude in Section 5.4.

5.1 The MTC Global Constraint Definition

We define a new constraint that encodes a single Markov transition.

Definition 5.1.1 (The Markov transition constraint)**.** Let vector

$$\mathbf{X} = [X_1, \ldots, X_N] \tag{5.4}$$

and vector

$$\mathbf{Y} = [Y_1, \ldots, Y_N] \tag{5.5}$$

be two vectors of variables that represent probability distributions over $\mathcal{N}$. The domains of the variables in vectors $\mathbf{X}$ and $\mathbf{Y}$ are:

$$\operatorname{dom}(X_i) = \left[\underline{X}_i, \overline{X}_i\right], \qquad \forall i \in \mathcal{N}, \tag{5.6}$$

where $\underline{X}_i$ and $\overline{X}_i$ are lower and upper bounds on variable X_i; and

$$\operatorname{dom}(Y_j) = \left[\underline{Y}_j, \overline{Y}_j\right], \qquad \forall j \in \mathcal{N}, \tag{5.7}$$

where $\underline{Y}_j$ and $\overline{Y}_j$ are lower and upper bounds on variable Y_j. Given $\mathbf{M}$, a known Markovian transition matrix, the Mtc is defined as follows:

$$\begin{aligned} &\textsc{Mtc}([Y_1, \ldots, Y_N], [X_1, \ldots, X_N], \mathbf{M}) \\ \Leftrightarrow& \\ &\forall j \in \mathcal{N} : \sum_{i \in \mathcal{N}} X_i \mathbf{M}_{ij} = Y_j \quad \wedge \quad \sum_{i \in \mathcal{N}} X_i = 1. \end{aligned} \tag{5.8}$$

The constraint $\textsc{Mtc}(\mathbf{Y}, \mathbf{X}, \mathbf{M})$ applies a transition matrix $\mathbf{M}$ to $\mathbf{X}$ and computes $\mathbf{Y}$, i.e., $\textsc{Mtc}(\mathbf{Y}, \mathbf{X}, \mathbf{M})$ states that $\mathbf{Y} = \mathbf{XM}$ while maintaining $\sum_{i \in \mathcal{N}} X_i = 1$. $\triangleleft$

Remark (Chaining Mtcs). Multiple Mtc constraints may be chained to compute a finite Markov chain of T steps

$$\textsc{Mtc}(\mathbf{X}^2, \mathbf{X}^1, \mathbf{M}), \textsc{Mtc}(\mathbf{X}^3, \mathbf{X}^2, \mathbf{M}), \ldots, \textsc{Mtc}(\mathbf{X}^T, \mathbf{X}^{T-1}, \mathbf{M}), \tag{5.9}$$

where vectors $\mathbf{X}^t$ (for $t \in \{2, \ldots, T\}$) are vectors of CP variables representing probabilities. This leads to the following constraint set to add to the model:

$$\textsc{Mtc}(\mathbf{X}^t, \mathbf{X}^{t-1}, \mathbf{M}), \qquad \forall t \{2, \ldots, T\}. \tag{5.10}$$

Other constraints may be added on the X_i variables to interact with the chain. $\triangleleft$

5.2 Domain Filtering for Markov Transitions

Since the constraint $\textsc{Mtc}(\mathbf{Y}, \mathbf{X}, \mathbf{M})$ is satisfied only if $\mathbf{X}$ and $\mathbf{Y}$ represent probability distributions, filtering this constraint can both be seen as an application of the *theory of interval-probability* and as a linear optimization problem [de Campos et al., 1994, Weichselberger, 2000].

Remark (Uncertain distributions in CP). In our CP context, the uncertain probability distributions arise from the domain of the variables. The vector

$$[\mathrm{dom}(X_1), \ldots, \mathrm{dom}(X_N)] = \left[[\underline{X}_1, \overline{X}_1] \ldots, [\underline{X}_N, \overline{X}_N]\right] \tag{5.11}$$

is the vector of the domains of the variables representing probabilities. The first condition to be an uncertain distribution is to have at least one of the variables that is not yet assigned. That is, there must exists one k such that $\underline{X}_k < \overline{X}_k$. The second condition is to be a distribution. That is, there must exists an assignment to the variables X_i that sums up to 1. Such an assignment exists if and only if

$$\sum_{i \in \mathcal{N}} \underline{X}_i \leq 1 \leq \sum_{i \in \mathcal{N}} \overline{X}_i. \tag{5.12}$$

The same remark holds for vector $\mathbf{Y}$. $\triangleleft$

We present three filtering algorithms to filter the variables X_i and Y_j ($\forall i, j \in \mathcal{N}$) subject to an Mtc constraint. The first algorithm, denoted MTC-IA, uses the interval arithmetic [Moore, 1966] that is applied on a decomposition of the constraint. Following [Weichselberger, 2000] who mentions that interval probability problems are linear optimization problems, the second algorithm, denoted MTC-LP, performs a linear optimization to achieve bounds consistency. The third algorithm, denoted MTC-FK, is a compromise between the two previous approaches. It relaxes the problem into a set of fractional knapsack constraints on which it enforces bounds consistency.

5.2.1 Interval Arithmetic Filtering

We decompose the global constraint Mtc($\mathbf{Y}, \mathbf{X}, \mathbf{M}$) into linear constraints as follows:

$$\sum_{i\in\mathcal{N}} X_i \mathbf{M}_{ij} = Y_j, \qquad \forall j \in \mathcal{N}, \tag{5.13}$$

$$\sum_{i\in\mathcal{N}} X_i = 1, \tag{5.14}$$

$$\sum_{j\in\mathcal{N}} Y_j {\mathbf{M}^{-1}}_{ij} = X_i, \qquad \forall i \in \mathcal{N}, \tag{5.15}$$

$$\sum_{j\in\mathcal{N}} Y_j = 1. \tag{5.16}$$

Constraints (5.13) and (5.14) follow from the definition of the Mtc. Constraints (5.15) and (5.16) are implied constraints that improve the filtering. We call MTC-IA the algorithm that uses interval arithmetic to enforce bounds consistency on the constraints (5.13) to (5.16). This algorithm, which is already implemented in most constraint solvers, simply enforces bounds consistency on each constraint individually until a fixed point is reached. The implied constraints necessitate the inverse of the transition matrix $\mathbf{M}$. The inverse $\mathbf{M}^{-1}$ is computed during the generation of the model, prior to solving the problem. This pre-solving process is done in $O(n^3)$ where n is the size of matrix $\mathbf{M}$. Implied constraints, even though they might enhance filtering, are not always used in CP models. We call MTC-IA- the algorithm that uses the interval arithmetic to enforce bounds consistency on the constraints (5.13) to (5.14).

We discuss specific cases where interval arithmetic enforces bounds consistency on the constraint Mtc. These cases require new definitions.

Definition 5.2.1 (Monomial matrix)**.** A matrix $\mathbf{A}$ is *monomial* if and only if it has one and only one non-null element in each column and each row. ◁

Proposition 1. *A monomial transition matrix* $\mathbf{M}$ *is a permutation matrix.*

Proof. The rows of $\mathbf{M}$ sums up to 1 by definition. Because $\mathbf{M}$ is monomial, we must have one element set to one in every row and all other elements are null which result into a binary matrix. Monomial binary matrices are permutation matrices. □

Proposition 2. *The inverse of a monomial transition matrix* $\mathbf{M}$ *is a transition matrix.*

Proof. $\mathbf{M}$ is a permutation matrix. The inverse of a permutation matrix is its transpose which is also a permutation matrix. □

Lemma 1. *If* $\mathbf{M}$ *is monomial, then enforcing bounds consistency on the linear constraints* (5.13) *and* (5.14) *enforces bounds consistency on* $\textsc{Mtc}(\mathbf{Y}, \mathbf{X}, \mathbf{M})$.

Proof. If $\mathbf{M}$ is monomial then it is a permutation matrix and the constraints (5.13) are binary equalities. Suppose that constraints (5.13) and (5.14) are bounds consistent. Let x be an interval support for (5.14) then $y = x\mathbf{M}$ is a permutation of x and, thanks to the equality constraints (5.13), we have $y_i \in \text{dom}(Y_i)$. Consequently, the upper bound and lower bounds of the domains $\text{dom}(X_i)$ have an interval support for $\textsc{Mtc}(\mathbf{Y}, \mathbf{X}, \mathbf{M})$. Let π be the permutation encoded by $\mathbf{M}$. Since the bounds of $\text{dom}(Y_{\pi(i)})$ are equal to the bounds of $\text{dom}(X_i)$, the bounds of $\text{dom}(Y_i)$ also have an interval support for $\textsc{Mtc}(\mathbf{Y}, \mathbf{X}, \mathbf{M})$. □

Thanks to Proposition 2, Lemma 1 also holds when replacing constraint (5.13) by constraint (5.15) and/or constraint (5.14) by constraint (5.16).

5.2.2 Linear Programming Filtering

In this section, we describe our linear programming (LP) reformulation of the filtering problem for the Mtc global constraint. Even though LP has not been applied (to our knowledge) to the filtering of global constraints encoding a Markov chain, there exist examples in the literature (e.g., [Bessiere et al., 2005] and [Bessiere et al., 2006]) where LP is used to filter global constraints. These successes justify the application of the idea to Markov chains.

The LP filtering algorithm solves two linear programs per variable: one for the lower bound and one for the upper bound. To obtain a lower bound on variable X_k, the LP minimizes X_k (equation (5.17)) subject to the constraints (5.18) to (5.21).

$$\min X_k \tag{5.17}$$

subject to

$$\sum_{i \in \mathcal{N}} \mathbf{M}_{ij} X_i = Y_j, \qquad \forall j \in \mathcal{N}, \tag{5.18}$$

$$\sum_{i \in \mathcal{N}} X_i = 1, \tag{5.19}$$

$$\underline{X}_i \leq X_i \leq \overline{X}_i, \qquad \forall i \in \mathcal{N}, \tag{5.20}$$

$$\underline{Y}_j \leq Y_j \leq \overline{Y}_j, \qquad \forall j \in \mathcal{N}. \tag{5.21}$$

New bounds on $\overline{X}_k$, $\underline{Y}_k$, and $\overline{Y}_k$ follow from a modification of the objective function. For each state $k \in \mathcal{N}$, we have the following LPs:

- $\overline{X}_k = \max X_k$ (resp. $\overline{Y}_k = \max Y_k$) subject to constraints (5.18) to (5.21);
- $\underline{X}_k = \min X_k$ (resp. $\underline{Y}_k = \min Y_k$) subject to constraints (5.18) to (5.21).

Each of the $4N$ linear programs may be solved using the simplex method [Dantzig, 1949]. We call this filtering technique based on linear optimization MTC-LP. If an exact LP method is used (e.g., the simplex algorithm), we obtain optimal bounds on the domains of both X and Y.

Theorem 5.2.1. *MTC-LP enforces bounds consistency on* $\textsc{Mtc}(\mathbf{Y}, \mathbf{X}, \mathbf{M})$.

Proof. The proof is a direct consequence of using an exact method for solving the linear programs. □

5.2.3 Fractional Knapsack Filtering

The last filtering algorithm we present, denoted MTC-FK, is based on the fractional knapsack problem and improves on the filtering done by the interval arithmetic. It is inspired from the global knapsack constraint [Katriel et al., 2007]. We consider, for some $l \in \mathcal{N}$, the pair of constraints $\sum_{i \in \mathcal{N}} X_i = 1$ and $\sum_{i \in \mathcal{N}} \mathbf{M}_{il} X_i = Y_l$. To compute an upper bound on the variable Y_l, one greedily assigns the largest possible values to the variables X_i that are multiplied by the greatest weights $\mathbf{M}_{il}$ while making sure that the constraint (5.14) is satisfied. To compute a lower bound on the variable Y_l, one needs to assign the largest values to the variables multiplied by the smallest weights.

Algorithm 2 propagates the constraints (5.13) to (5.16) as well as the knapsack constraints

$$\sum_{i \in \mathcal{N}} X_i \mathbf{M}_{ij} = Y_j, \quad \sum_{i \in \mathcal{N}} X_i = 1, \tag{5.22}$$

and

$$\sum_{j \in \mathcal{N}} Y_j {\mathbf{M}^{-1}}_{ij} = X_i, \quad \sum_{j \in \mathcal{N}} Y_j = 1 \tag{5.23}$$

until it reaches a precision of ϵ. Algorithm 3 compute a lower bound on Y_l (or X_l). Reversing the order of the iterations in the for loop makes Algorithm 3 computes an upper bound on Y_l (or X_l).

Lemma 2. *Let $\overline{m}_j$ and $\underline{m}_j$ be the greatest and smallest value in column j of the matrix* $\mathbf{M}$. *If $\left[\underline{m}_j, \overline{m}_j\right] \subseteq dom(Y_j)$ $\forall j \in \mathcal{N}$ then any distribution $x_1, \ldots, x_N$ (i.e., any assignment to the variables of vector* $\mathbf{X}$ *that sums to one) has a support in* $\textsc{Mtc}(\mathbf{Y}, \mathbf{X}, \mathbf{M})$.

Proof. Let $\mathbf{M}^j$ be the j^{th} column of $\mathbf{M}$. Since the components of $\mathbf{X}$ sum to one and since none are negative, the scalar product of $\mathbf{X}$ and $\mathbf{M}^j$ is a convex combination of the elements

```
Function MTC-FK($[X_1,\dots,X_N],[Y_1,\dots,Y_N],\mathbf{M},\mathbf{M}^{-1}$)
    Input: A vector of variables that represents the current uncertain distribution over $\mathcal{N}$:
           $[X_1,\dots,X_N]$; a vector of variables that represents the resulting uncertain
           distribution: $[Y_1,\dots,Y_N]$; a transition matrix and its inverse: $\mathbf{M}$ and $\mathbf{M}^{-1}$.
    Output: The vectors of probability variables with filtered domain: $[X_1,\dots,X_N]$, and
            $[Y_1,\dots,Y_N]$.
    repeat
        for $i \in \mathcal{N}$ do $x_i^{\text{old}} \leftarrow \overline{X}_i - \underline{X}_i$;
        for $i \in \mathcal{N}$ do $y_i^{\text{old}} \leftarrow \overline{Y}_i - \underline{Y}_i$;
        Enforce bounds consistency on constraints (5.13) to (5.16);
        foreach $l \in \mathcal{N}$ do
            $\underline{Y}_l \leftarrow$ Fk-FilterLowerBound ($[X_1,\dots,X_N]$,$Y_l$,$[\mathbf{M}_{1l},\dots,\mathbf{M}_{Nl}]$);
            $\overline{Y}_l \leftarrow$ Fk-FilterUpperBound ($[X_1,\dots,X_N]$,$Y_l$,$[\mathbf{M}_{1l},\dots,\mathbf{M}_{Nl}]$);
            $\underline{X}_l \leftarrow$ Fk-FilterLowerBound ($[Y_1,\dots,Y_N]$,$X_l$,$[\mathbf{M}^{-1}_{1l},\dots,\mathbf{M}^{-1}_{Nl}]$);
            $\overline{X}_l \leftarrow$ Fk-FilterUpperBound ($[Y_1,\dots,Y_N]$,$X_l$,$[\mathbf{M}^{-1}_{1l},\dots,\mathbf{M}^{-1}_{Nl}]$);
    until $\sqrt{\sum_{i\in\mathcal{N}}(x_i^{old} - \overline{X}_i + \underline{X}_i)^2} + \sqrt{\sum_{i\in\mathcal{N}}(y_i^{old} - \overline{Y}_i + \underline{Y}_i)^2} \le \epsilon$;
    return $[X_1,\dots,X_N]$, $[Y_1,\dots,Y_N]$;
```

Algorithm 2: The MTC-FK filtering algorithm

```
Function FK-FilterLowerBound($[U_1,\dots,U_N]$,$\underline{V}_l$, $[t_1,\dots,t_N]$)
    Input: A vector of variables that represents an uncertain distribution over $\mathcal{N}$:
           $[U_1,\dots,U_N]$; a lower bound on the variable that represents the probability that
           the process is in state $l$ after applying transitions: $\underline{V}_l$; a vector of the transition
           probabilities to state $l$: $[t_1,\dots,t_N]$.
    Output: A new lower bound on state $l$ probability: $\underline{V}_l$.
    $\lambda \leftarrow 1 - \sum_{l\in\mathcal{N}} \underline{U}_l$;
    for $k \in \mathcal{N}$ in non-decreasing order of $t_k$ do
        $\delta \leftarrow \min\left(\lambda, \overline{U}_k - \underline{U}_k\right)$;
        $u_k \leftarrow \underline{U}_k + \delta$;
        $\lambda \leftarrow \lambda - \delta$;
        if $\lambda = 0$ then break;
    return $\max\left(\sum_{i\in\mathcal{N}} u_i t_i, \underline{V}_l\right)$;
```

Algorithm 3: The FK-FilterLowerBound lower bound filtering algorithm

in $\mathbf{M}^j$. Consequently, the result lies in the convex hull of $\mathbf{M}^j$ and it cannot be greater nor smaller than any element in $\mathbf{M}^j$. $\square$

Lemma 3. *Let $\overline{m}_j$ and $\underline{m}_j$ be the greatest and smallest value in column j of the matrix $\mathbf{M}$. If $\left[\underline{m}_j, \overline{m}_j\right] \subseteq dom(Y_j)$ $\forall j \in \mathcal{N}$ then MTC-FK enforces bounds consistency on* $\text{Mtc}(\mathbf{Y}, \mathbf{X}, \mathbf{M})$.

Proof. The algorithm MTC-FK enforces bounds consistency on the constraint $\sum_{i \in \mathcal{N}} X_i = 1$ so that the lower bounds and upper bounds of the domains of the X_i can each form an assignment of the variables X_i that sums to one. From Lemma 2, any assignment that sums to 1 can be extended to a support of $\text{Mtc}(\mathbf{Y}, \mathbf{X}, \mathbf{M})$. We now need to prove that the variables Y_i are fully pruned. If a bound of $\text{dom}(Y_i)$ is modified, this bound was computed by the Algorithm 3 which constructed a valid support for the constraint. If a bound of $\text{dom}(Y_i)$ is not filtered, then either Algorithm 3 computed the same bound (in which case, it has a support) or it computed a larger one. However, the second case cannot occur since, as seen in Lemma 2, the scalar product of any distribution with a column of $\mathbf{M}$ leads to a value in $\left[\underline{m}_j, \overline{m}_j\right] \subseteq \text{dom}(Y_j)$. $\square$

Lemma 3 is particularly useful at the beginning of the search in a problem where the domains of the variables Y_i are the intervals $[0, 1]$.

Theorem 5.2.2. *The consistency achieved by each algorithm satisfies MTC-IA $\prec$ MTC-FK $\prec$ MTC-LP.*

Proof. We first prove MTC-IA $\preceq$ MTC-FK $\preceq$ MTC-LP. The MTC-FK algorithm filters the same constraints as MTC-IA but the knapsack constraints consider pairs of constraints which offer a filtering that is not weaker. The algorithm MTC-LP achieves bounds consistency which is optimal. We now show two examples which prove that MTC-IA $\neq$ MTC-FK, and that MTC-FK $\neq$ MTC-LP. Let $N = 3$. Let $\text{dom}(X_1) = [.3, 1]$, and $\text{dom}(X_2) = \text{dom}(X_3) = [0, 1]$. Let $\text{dom}(Y_1) = \text{dom}(Y_2) = \text{dom}(Y_3) = [0, 1]$. Let the transition matrix be

$$\mathbf{M} = \begin{bmatrix} 0 & .4 & .6 \\ .3 & .4 & .3 \\ .4 & .6 & 0 \end{bmatrix}. \tag{5.24}$$

By Lemma 2, MTC-FK enforces bounds consistency whereas MTC-IA does not. In fact, MTC-FK sets $\overline{Y}_1 = 0.28$ whereas MTC-IA sets $\overline{Y}_1 = 0.49$. Suppose that $\text{dom}(Y_1) = [.1, 1]$. By Theorem 5.2.1, MTC-LP enforces bounds consistency which is not the case of MTC-FK. In fact, MTC-LP sets $\overline{X}_1 = 0.75$ while MTC-FK sets $\overline{X}_1 = 0.9$. $\square$

5.2.4 Experiments on the Filtering of a Single MTC

In this section, we present empirical results to compare our filtering algorithms for the Mtc. Each filtering algorithm was implemented in Octave 3.6.4 [Eaton, 2013] (Matlab 7.10.0.499

Table 5.1: The characteristics of the instance sets

Name	N	$\mathbf{M}$	$\rho = \mathbf{M}_{ii}$	Set size
Random	$\{2, \ldots, 100\}$	Random	Random	990
Star grids	$\{4, 9 \ldots, 100\}$	Star grids	$\{.2, .4, .6, .8\}$	360
Plus grids	$\{4, 9, \ldots, 100\}$	Plus grids	$\{.2, .4, .6, .8\}$	360
Zero-one-Y	$\{2, \ldots, 100\}$	Random	Random	990
Zero-one-X	$\{2, \ldots, 100\}$	Random	Random	990

(R2010a) [Mathworks, 2010]). We used the GLPK solver [Makhorin, 2012] and the IBM ILOG CPLEX 12.5 [IBM, 2013] solver to solve our linear programs for the MTC-LP filtering algorithm. We tend to prefer GLPK for this particular problem as it is lightweight. However, CPLEX benefits from a higher numerical stability. Since Octave (Matlab) is an interpreted language, we would add that the purpose of these experiments is not to compare the algorithms on their processing speed, but rather to have insights on the quality their filtering.

We generated random domains and transition matrices. Each pair of random domains and transition matrix is a single instance of the problem of filtering a single Mtc. For MTC-IA, we present the results with and without the implied constraints (5.15) and (5.16), an algorithm we call MTC-IA-. The MTC-LP enforces bounds consistency on all instances thus providing the optimal bounds. Let $\underline{\mathbf{x}}^{\text{old}}$, $\overline{\mathbf{x}}^{\text{old}}$, $\underline{\mathbf{y}}^{\text{old}}$ and $\overline{\mathbf{y}}^{\text{old}}$ be the initial bounds of the filtering problem. Let $\underline{\mathbf{x}}^{*}$, $\overline{\mathbf{x}}^{*}$, $\underline{\mathbf{y}}^{*}$ and $\overline{\mathbf{y}}^{*}$ be the optimal bounds found by MTC-LP. Let $\underline{\mathbf{x}}$, $\overline{\mathbf{x}}$, $\underline{\mathbf{y}}$ and $\overline{\mathbf{y}}$ be the bounds found by a given filtering method. We define the *proportion of optimality* as the ratio of the sum of the distances traveled, in the domain space, by a filtering method to the sum of the distances traveled by MTC-LP:

$$Ind_{\text{p}} = \frac{\left\|\underline{\mathbf{x}} - \underline{\mathbf{x}}^{\text{old}}\right\| + \left\|\overline{\mathbf{x}}^{\text{old}} - \overline{\mathbf{x}}\right\| + \left\|\underline{\mathbf{y}} - \underline{\mathbf{y}}^{\text{old}}\right\| + \left\|\overline{\mathbf{y}}^{\text{old}} - \overline{\mathbf{y}}\right\|}{\|\underline{\mathbf{x}}^{*} - \underline{\mathbf{x}}^{\text{old}}\| + \|\overline{\mathbf{x}}^{\text{old}} - \overline{\mathbf{x}}^{*}\| + \left\|\underline{\mathbf{y}}^{*} - \underline{\mathbf{y}}^{\text{old}}\right\| + \|\overline{\mathbf{y}}^{\text{old}} - \overline{\mathbf{y}}^{*}\|}. \tag{5.25}$$

Filtering problem instances for which $\underline{\mathbf{x}} = \underline{\mathbf{x}}^{*}$, $\overline{\mathbf{x}} = \overline{\mathbf{x}}^{*}$, $\underline{\mathbf{y}} = \underline{\mathbf{y}}^{*}$, and $\overline{\mathbf{y}} = \overline{\mathbf{y}}^{*}$ is obviously not interesting since they are already solved to optimality.

We generated five sets of filtering problem instances: a transition matrix $\mathbf{M}$ along with random bounds on the variables of vectors $\mathbf{X}$ and $\mathbf{Y}$. The randomly generated bounds are feasible, i.e., the constraint Mtc is satisfiable. The first set (*random*) contains randomly generated transition matrices. The second and the third sets (*star* and *plus grids*) are made of square grids. For these matrices, each state is located in a cell of the grid. A state is connected to its neighbors as follows:

- in the *Plus grids* instance set, each state is linked to its North, South, West and East neighbors;

- in the *Star grids* instance set, each state is linked to its North, South, West and East neighbors plus to its diagonals neighbors (NW, NE, SW, SE).

The transition matrix of a grid instance follows an OSP-like motion model. Let ρ be the conditional probability that the process stays in state i when it is in state i for any $i \in \mathcal{N}$, i.e., the *probability of stationarity* $\rho = \mathbf{M}_{ii}$ $(\forall i \in \mathcal{N})$. Let $\deg(i)$ be the degree of cell i (loops included) in the adjacency matrix of the grid. The transition matrix of a grid instance is defined as:

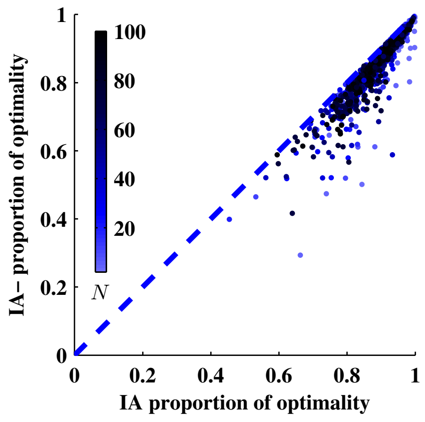
$$\mathbf{M}_{ij} = \begin{cases} \frac{1-\rho}{\deg(i)-1} & \text{if } i \neq j; \\ \rho & \text{if } i = j. \end{cases} \tag{5.26}$$

We chose $\rho \in \{.2, .4, .6, .8\}$. The fourth set (*zero-one-Y*) contains randomly generated transition matrices, but the domains of the variables of vector $\mathbf{Y}$ are $[0, 1]$. That is, the information about the future (i.e., the uncertain distribution of vector $\mathbf{Y}$) comes from the present (i.e., the uncertain distribution of vector $\mathbf{X}$). This is the usual way to process Markov chains in Markov processes. The fifth set (*zero-one-X*) models the converse case where the information about the present state of the process comes from the future. Our benchmark library includes non-singular matrices only. We generated 10 different instances for each pair of state space size N and probability of stationarity ρ in each set for a total of 3690 instances. Table 5.1 summarizes the characteristics of the sets.

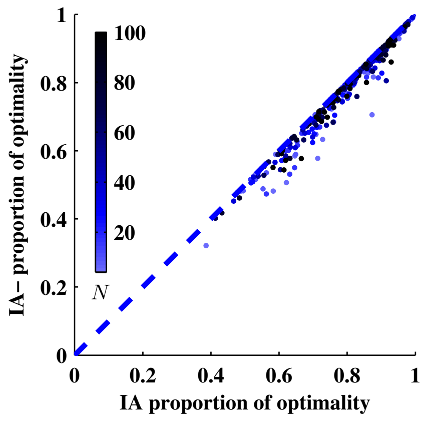
5.2.5 Results and Discussion

Figures 5.1 to 5.3 show scatter plots comparing the proportion of optimality achieved by different filtering algorithms. The higher the proportion of optimality is on a given axis, the better the filtering algorithm performs. A value of 1 represents a bounds consistent domain as obtained by the MTC-LP algorithm. The MTC-LP algorithm achieved bounds consistency in all cases (either using GLPK or CPLEX). It is thus a consistent point of comparison for the other algorithms. The dotted line is a visual frontier between the two compared algorithms' performance. Dots on this visual frontier belong to instances for which both compared algorithms produce the same filtering. A dot for which the algorithm on the x-axis (y-axis) performs better than the algorithm on the y-axis (x-axis) lies on the right (left) hand-side of the frontier. The algorithm with the highest density of dots on its side of the frontier tends to achieve the best overall performance. Darker blue shades are used for instances with a larger state space size (N); lighter blue shades are used for instances with a smaller N.

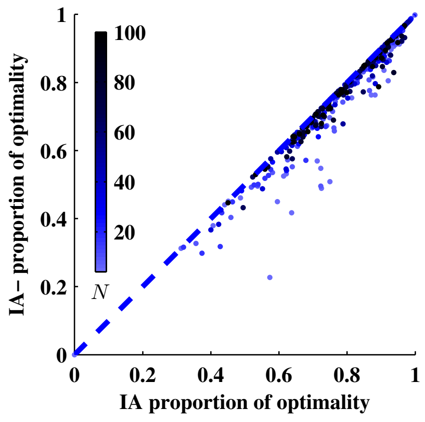
As shown on Figure 5.1, MTC-IA (x-axis) outperforms MTC-IA- (i.e., MTC-IA without implied constraints) (y-axis) on all instance sets. While the performance of the two algorithms is similar on some random instances (Figure 5.1(a)), the importance of implied constraints



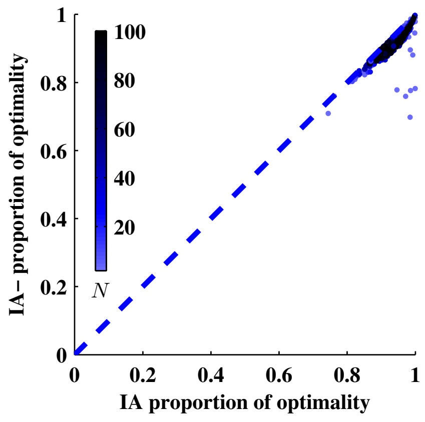
(a) Random instances



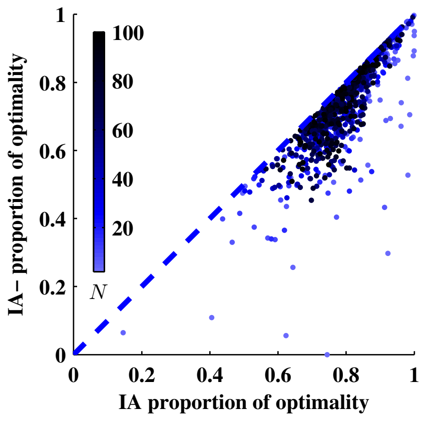
(b) Plus grids instances



(c) Star grids instances

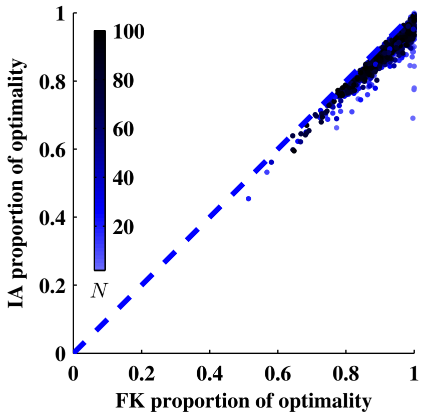


(d) Zero-one-Y instances

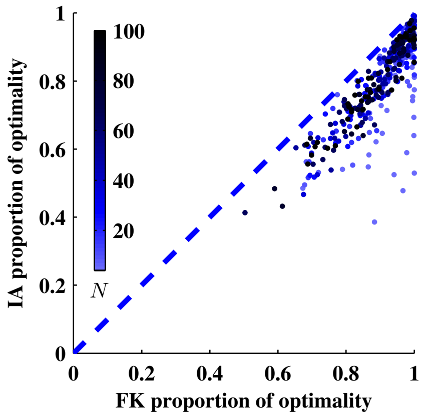


(e) Zero-one-X instances

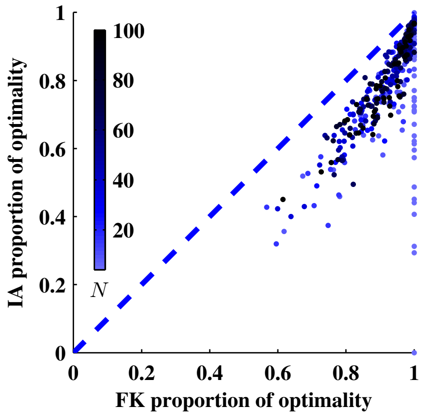
Figure 5.1: Proportion of optimality achieved by MTC-IA (IA) when compared to MTC-IA without implied constraints (IA-)



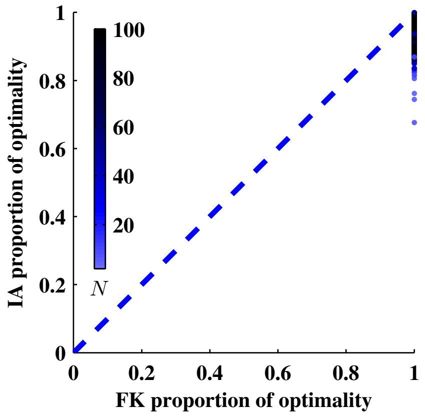
(a) Random instances



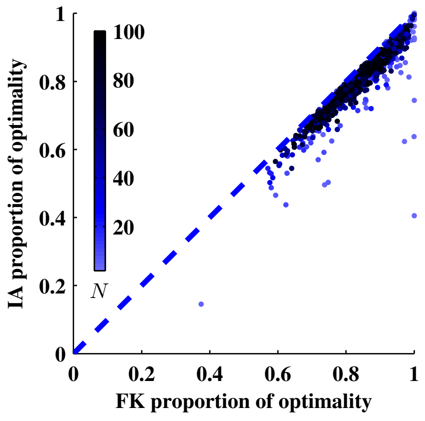
(b) Plus grids instances



(c) Star grids instances

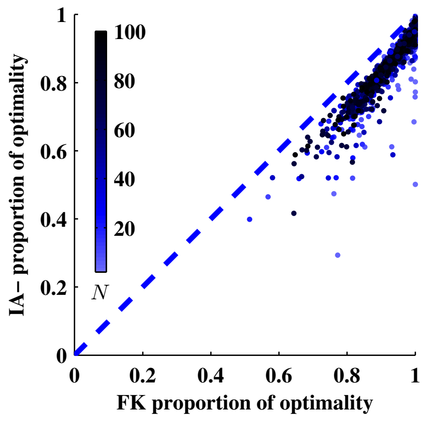


(d) Zero-one-Y instances

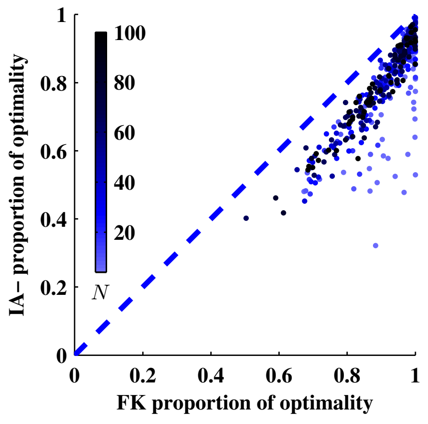


(e) Zero-one-X instances

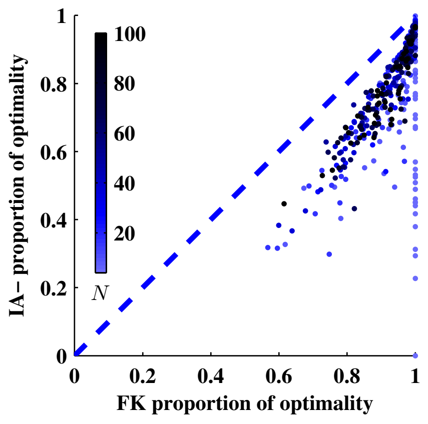
Figure 5.2: Proportion of optimality achieved by MTC-FK (FK) when compared to MTC-IA (IA)



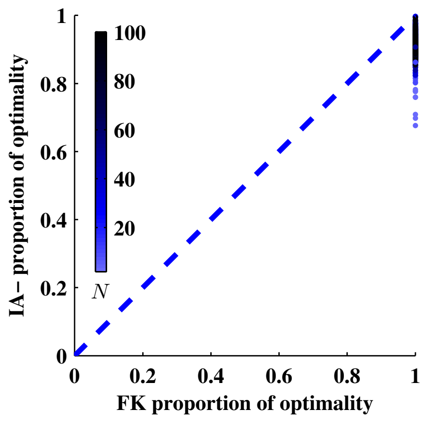
(a) Random instances



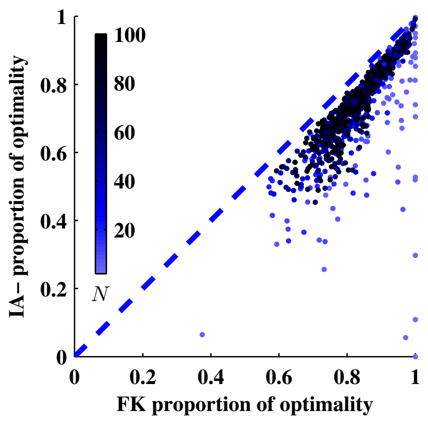
(b) Plus grids instances



(c) Star grids instances



(d) Zero-one-Y instances



(e) Zero-one-X instances

Figure 5.3: Proportion of optimality achieved by MTC-FK (FK) when compared to MTC-IA without implied constraints (IA-)

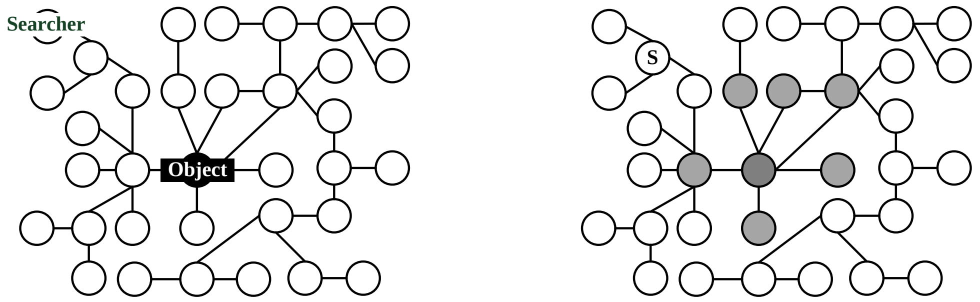
is clear as all the dots fall on the right hand side of the frontier. The scatter plots of the grid instance sets (Figures 5.1(b) and 5.1(c)) favors MTC-IA. Zero-one-Y instances represent forward in time inferences using $\mathbf{M}$ (Figure 5.1(d)). Zero-one-X instances represent backward in time inferences using $\mathbf{M}^{-1}$ (Figure 5.1(e)). On these instances, we see the benefits of the interaction between a set of elementary constraints: the implied constraints enable the algorithm to further filter the domains backward in time whenever knowledge on the future is acquired by forward filtering. The difficulty of backward inferences is shown by the fact that the distribution of the results on the Zero-one-Y instances (forward inference) is closer to 1 when compared to the distribution of the results on the Zero-one-X instances (backward inference). This is partly due to the negative values in the inverse of most transition matrices.

As shown on Figure 5.2, MTC-FK (x-axis) outperforms MTC-IA (y-axis) on all instance sets. The performance of both algorithms is close on the Random set (Figure 5.2(a)), but still, MTC-FK performs better. It is clear that MTC-FK outperforms MTC-IA on the grid instances (Figures 5.2(b) and 5.2(c)). We see that the increase of performance is mostly due to MTC-FK by comparing these results to the ones of Figures 5.1(b) and 5.1(c) that show how close the performance of MTC-IA is from the one of MTC-IA- on grids. Furthermore, MTC-FK enforced bounds consistency on all Zero-one-Y instances (a result of Lemma 3) whereas this is not the case for MTC-IA. Forward inference is easier than backward inference for both algorithms: the distribution of the optimality results is closer to 1 on the Zero-one-Y set than on the Zero-one-X set.

We recall that MTC-IA- is the intuitive way of decomposing a Markov chain in CP. That is, without the use of any implied constraint. We present, in Figure 5.3, the scatter plots comparing the proportion of optimality achieved by the filtering obtained with MTC-FK to this usual method. By comparing the scatter plots of 5.2 to the ones of 5.3, we clearly see how implied constraints can help MTC-IA- in improving its performance. This is especially true in the case of the hardest instance sets Random and Zero-one-X (resp. Figures 5.3(a) and 5.3(e)) where the distribution of the results tends to be farther of the frontier compared to the one obtained when comparing MTC-FK to MTC-IA. This comparison favor MTC-FK. Its efficiency is clear when compared to the usual way of modeling Markov chains (MTC-IA-).

5.3 Application to Search Path Planning

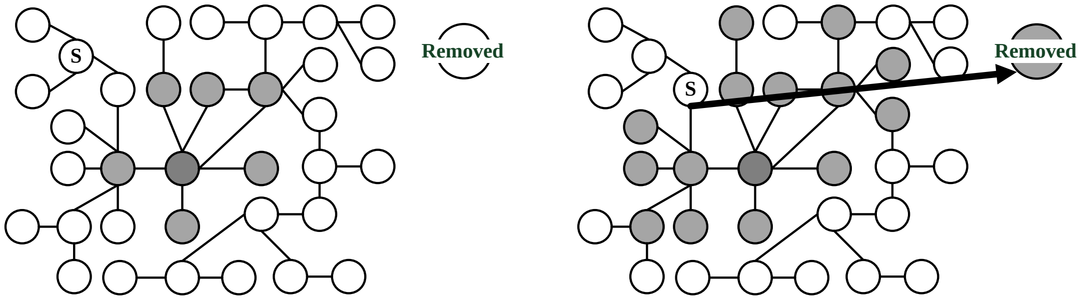
We illustrate the Mtc constraint usage on an *optimal search path* (OSP) problem as defined in Chapter 2, the same problem we tackled in Chapter 4 using the TD heuristic and CP. We first formulate the problem using a split Markov chain to allow us to use Mtcs in the OSP definition. Then we present the CP model. We say that the Markov chain is *split* since events that influence the probability distribution over the state space occur between the transitions.



(a) At time $t = 1$, the searcher is in the upper left corner and the object is in the middle vertex

(b) At time $t = 2$, the searcher and the object move to an adjacent vertex; the object follows its motion model defined by matrix $\mathbf{M}$

Figure 5.4: A search environment with a one time step motion without search



(a) At time $t = 2$, the searcher searches its current vertex; the containment probability is null in that vertex

(b) At time $t = 3$, the searcher moves to an adjacent location and searches again; the vertex has a non-null containment probability and some probability mass is displaced to the "removed" state

Figure 5.5: A search environment with a one time step motion with a search

5.3.1 The OSP as a split Markov chain

A searcher moves on a graph $G_A = (\mathcal{V}(G_A), \mathcal{E}(G_A))$ in order to find a lost object within T time steps. In absence of search, the object moves from vertex to vertex according to a known transition matrix $\mathbf{M}$ (Figure 5.4). The probability that a vertex r contains the object at a time t is called the *probability of containment*, or $poc_t(r)$. The initial distribution, i.e., poc_1, is known a priori. The searcher influences the evolution of the chain by searching the vertices and by removing the object as soon as s/he detects it. A special "removed" state that represents the searcher's hands is added to the state space to take searches into account (Figure 5.5). The probability mass in the "removed" state corresponds to the *cumulative overall probability of success*, i.e., to the OSP objective function we presented in Sections 2.2.2 and 2.2.3 of Chapter 2 (see also [Frost, 1999d, Stone, 2004]).

We redefine the problem more formally as a split Markov chain. The distribution $\mathbf{x}_t$ over the states is

$$\mathbf{x}_t = \left[poc_t(1), \ldots, poc_t(N), cos_{t-1}\right], \tag{5.27}$$

where cos_t, the searcher's *cumulative overall probability of success* at time t, corresponds to the probability mass in the removed state at time t (with $cos_0 = 0$). When the searcher is located in a given vertex r at time t, i.e., $y_t = r$, s/he searches that vertex. Her/his known probability of detection in a vertex r, i.e., $pod(r)$, is conditional to the object's presence (and assumed independent of past searches). At a time t, the searcher sees her/his current location only. The local success of the searcher in r at time t is thus:

$$pos_t(r) = \begin{cases} poc_t(r) \times pod(r) & \text{if } r = y_t, \\ 0 & \text{if } r \neq y_t. \end{cases} \tag{5.28}$$

This local success probability at time t is the probability mass that we move to the removed state at that time step. We recall that the success up to time t, i.e., the probability that the object is in the removed state, is

$$cos_t = \begin{cases} pos_t(y_t) & \text{if } t = 1; \\ cos_{t-1} + pos_t(y_t) & \text{otherwise.} \end{cases} \tag{5.29}$$

We split the chain by introducing a distribution $\widehat{\mathbf{x}}^t$ that models the search:

$$\widehat{\mathbf{x}}^t = \left[poc_t(1) - pos_t(1), \ldots, poc_t(N) - pos_t(N), cos_t\right]. \tag{5.30}$$

Finally, we apply the object's motion to the searched distribution as follows:

$$\mathbf{x}^{t+1} = \widehat{\mathbf{x}}^t \begin{bmatrix} \mathbf{M} & 0 \\ 0 & 1 \end{bmatrix}. \tag{5.31}$$

The searcher's goal is to maximize cos_T.

5.3.2 A CP model with Mtcs for the OSP

The OSP, as defined in the previous subsection, leads to a novel CP model with four sets of interval-domain probability variables:

- the variables $POS_t(r)$ ($\forall t \in \{1, \ldots, T\}, r \in \mathcal{V}(G_A)$) represent the local successes:
  - given t and r, the variable $POS_t(r)$ corresponds to the probability mass to remove from the vertex r of the graph at time t;
- the variables $POC_t(r)$ and $POC_t^{\text{search}}(r)$ model the split Markov chain ($\forall t \in \{1, \ldots, T\}$, $r \in \mathcal{V}(G_A)$):

  - given t and r, the variable $POC_t(r)$ is the probability of containment of the object in vertex r after a move and *before* the search action that occurs at time t;
  - given t and r, the variable $POC_t^{\text{search}}(r)$ is the probability of containment of the object in vertex r *after* the search action that occurs at time t;

- the variables COS_t represent the probability of finding the object up to time t ($\forall t \in \{1, \dots, T\}$).

A set of finite-domain variables $PATH_t$ models the searcher's path. The domains of each path variable is a subset of vertices, i.e., $\text{dom}(PATH_t) \subseteq \mathcal{V}(G_A)$ ($\forall t \in \{1, \dots, T\}$). $POC_1(r) = poc_1(r)$, and $PATH_0 = y_0$ (where y_0 can be used to constraint the initial position of the searcher) are known data. The probability of finding (removing) the object up to time t is:

$$COS_t = \sum_{1 \le t' \le t} \max_{r \in \mathcal{V}(G_A)} POS_{t'}(r). \tag{5.32}$$

The objective is to maximize COS_T subject to the graph edges constraints:

$$(PATH_{t-1}, PATH_t) \in \mathcal{E}(G_A), \qquad \forall t \in \{1, \dots, T\}; \tag{5.33}$$

the probabilities of success along the path of the searcher:

$$PATH_t = r \implies POS_t(r) = POC_t(r) pod(r), \qquad \forall t \in \{1, \dots, T\}, \forall r \in \mathcal{V}(G_A); \tag{5.34}$$

$$PATH_t \neq r \implies POS_t(r) = 0, \qquad \forall t \in \{1, \dots, T\}, \forall r \in \mathcal{V}(G_A); \tag{5.35}$$

the effect of searching on the chain:

$$POC_t^{\text{search}}(r) = POC_t(r) - POS_t(r), \qquad \forall t \in \{1, \dots, T-1\}, \forall r \in \mathcal{V}(G_A); \tag{5.36}$$

and the application of the Markovian transition matrix of the object on the split chain:

$$\textsc{Mtc}\left(\mathbf{X}^{t+1}, \widehat{\mathbf{X}}^t, \begin{bmatrix} \mathbf{M} & 0 \\ 0 & 1 \end{bmatrix}\right), \qquad \forall t \in \{1, \dots, T-1\}, \tag{5.37}$$

where

$$\mathbf{X}^{t+1} = [POC_{t+1}(1), \dots, POC_{t+1}(N), COS_t], \tag{5.38}$$

and

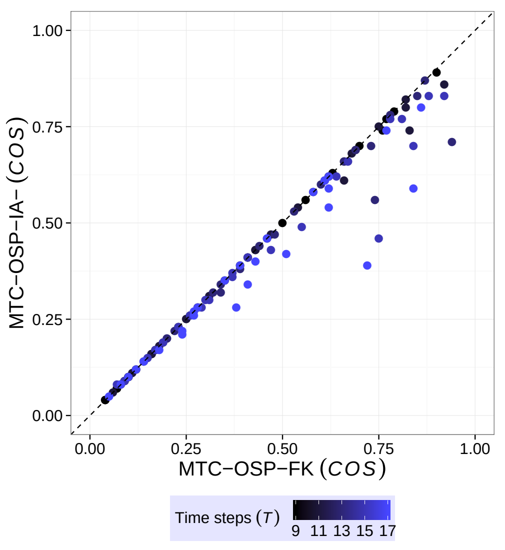
$$\widehat{\mathbf{X}}^t = \left[POC_t^{\text{search}}(1), \dots, POC_t^{\text{search}}(N), COS_t\right]. \tag{5.39}$$

5.3.3 Experiments on the OSP Problem

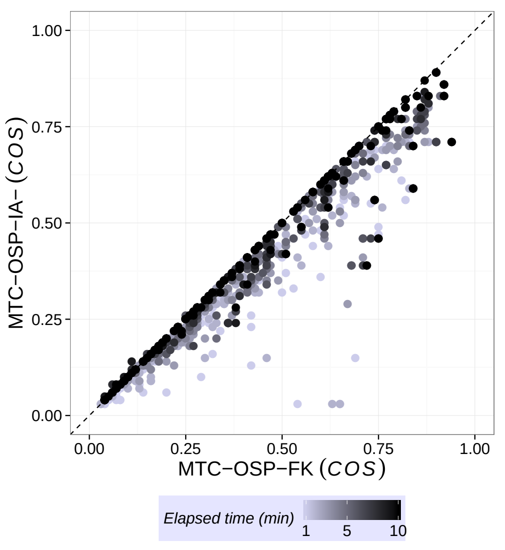
We implemented two Markov chain-based models for the OSP using Choco Solver 2.1.5 [Laburthe and Jussien, 2012]. The first, *MTC-OSP-IA-*, uses a standard elementary arithmetic constraints decomposition in the implementation of Constraint (5.37). This model is equivalent to the *OSP-MAX* model presented in Section 4.1 of Chapter 4. The second, *MTC-OSP-FK*, uses fractional knapsack filtering in the implementation of Constraint (5.37). We kept the model as close as possible to the one presented in Chapter 4 while adding the necessary variables to model the search with Mtcs. The goal of these experiments is to show the benefits of the additional filtering triggered by the Mtc when compared to the elementary arithmetic constraints decomposition. We first compare the results obtained by our chosen solver when using the *MTC-OSP-IA-* model against the ones obtained when using the *MTC-OSP-FK*. In both cases, the solver branches in the natural static order of the path variables (i.e., from $PATH_1$ to $PATH_T$) and then apply a decreasing domain heuristic to instantiate the next variable. Second, we discuss the results obtained when using both models along with the TD heuristic from Chapter 4. Again, the solver branches on the path variables in the order of the time steps, but it uses the TD heuristic to select the value from the domain of the variable to instantiate. We call these models *MTC-OSP-TD-IA-* and *MTC-OSP-TD-FK*.

We followed, for our experiments on the application of the Mtc global constraint to the OSP problem, the experimental framework presented in Section 4.1.1 of Chapter 4. The accessibility graphs (G_A) of the OSP problem instances in our benchmark are the following: G^+, a 11×11 plus grid where adjacent vertices are linked by an edge (diagonals excluded), G^* a 11×11 grid where adjacent vertices are linked by an edge (diagonals included), and the Université Laval tunnels map G^L which is the graph we used in the example of Figure 5.4(a) (see also Figure 4.1 in Section 4.1.1 of Chapter 4). The object starts in the middle of the graphs. The searcher starts in the upper left corner. The total number of time steps allowed for the searches are $T \in \{9, 11, 13, 15, 17\}$. For each problem instance, the detection probabilities ($pod(r)$) are uniform across regions. The detection probabilities are $pod(r) \in \{0.3, 0.6, 0.9\}$. The motion model of the search object, i.e., its transition matrix $\mathbf{M}$, is as defined by Equation (5.26). That is, there is a probability ρ that the object stays in place. The remaining probability mass is distributed uniformly on the accessible positions. We chose $\rho \in \{0.3, 0.6, 0.9\}$.

All implementations are done using the Java programming language, the Apache Commons Math library [Commons, 2010], the Java Universal Network/Graph (JUNG) 2.0.1 framework [O'Madadhain et al., 2010], and Choco Solver 2.1.5 [Laburthe and Jussien, 2012]. We found out that a precision of $\epsilon = 10^{-2}$ is sufficient for the Mtc to have a positive impact on the performance of the solver in our problem instances. It should however be understood that this value is dependent on the OSP problem instances to solve as well as on the precision used in the domain of the probability variables. As we did in the experiments presented in Chapter 4, we mapped the domain of the probability variables of our OSP CP models from



(a) *MTC-OSP-IA-* against *MTC-OSP-FK*: Last incumbent objective value within 10 minutes; the complexity of the instance (in terms of allowed time steps T) is represented by a gradient from black to blue.



(b) *MTC-OSP-IA-* against *MTC-OSP-FK*: Last incumbent objective value in time; the higher the allowed time is, the darker the dot is.

Figure 5.6: Comparison of the objective values obtained by the solver when using the *MTC-OSP-IA-* model to the objective values obtained by the solver when using the *MTC-OSP-FK* model; each dot is a comparison of the incumbent solutions found by the compared methods on a single OSP problem instance.

reals in $[0, 1]$ to integers in $[0, U]$ (with $U = 10^4$). This is required for implementation purposes in Choco 2.1.5. This leads to a precision of four decimals in computations. This is an implementation characteristic and a constraint of the chosen solver. It is not a limitation of the Mtc filtering algorithms which would be allowed to perform at their full potential when implemented in a solver allowing the use of real-valued variables with bounded domains of a higher decimal precision. Computations were made on the supercomputer Colosse from Université Laval. We allowed a total number of 5,000,000 backtracks and a time limit of 10 minutes.

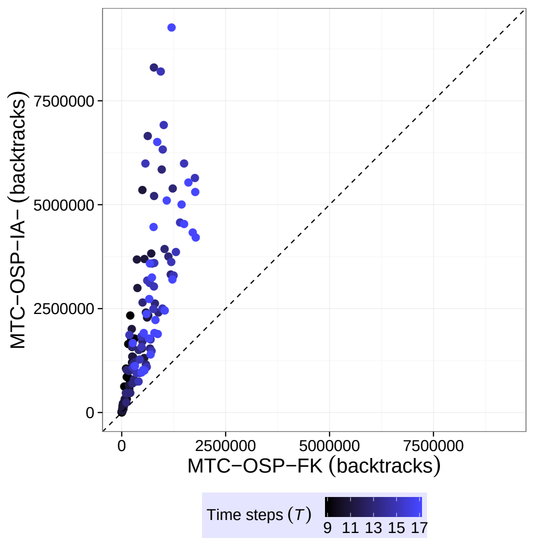
5.3.4 Results and Discussion

We compare, on Figure 5.6, the objective values obtained by the solver when using the *MTC-OSP-IA-* model against the objective values obtained by the solver when using the *MTC-OSP-FK* model. Each dot on the subfigure on the left-hand side (see Figure 5.6(a)) is a comparison of the objective value of two incumbent solutions of a single OSP problem instance. The first solution is the one found by the method on the y axis and the other is the one found by the method on the x axis. A dot on the dashed frontier represents a tie. A dot on the right-hand side (resp. left-hand side) of the frontier is a win for the method on the x axis (resp. y

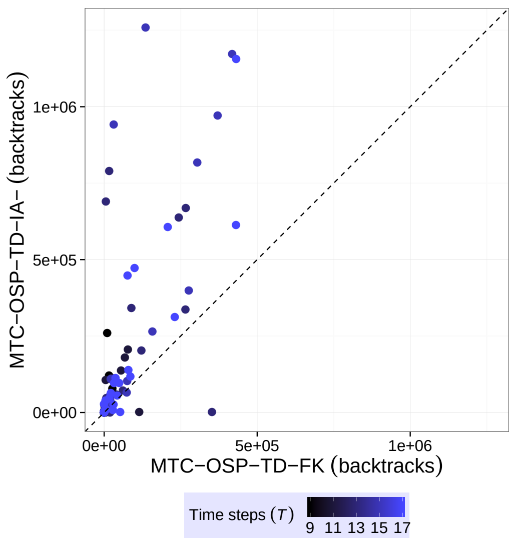
axis). The gradient from black to blue is used to represent the total number of allowed time steps (from $T = 9$, in black, to $T = 17$, in light blue). The subfigure on the right-hand side (see Figure 5.6(b)) represents a similar information, but at intervals of one minute from the beginning to the end of the solving process leading to a total of 10 snapshots in time, i.e., from one to ten minutes of allowed solving time. A gradient from gray (1 minute of solving time) to black (10 minutes of solving time) identifies to which snapshot a dot corresponds. The best so far objective value at each time interval can thus be compared. We see, from the figures, that the supplementary filtering triggered by the Mtc global constraint when using fractional knapsack filtering is beneficial from the beginning to the end of the solving process. Effectively, the vast majority of the solutions found by the solver at different intervals during the solving process are on the right-hand side of Figure 5.6(b) meaning that a more thorough filtering leads to an improved performance.

We present a comparison of the total number of backtracks triggered by elementary constraints filtering against the total number of backtracks triggered by fractional knapsack filtering on Figure 5.7. These are the number of backtracks required by both methods to find their best incumbent solution. A better or equal objective value plus a fewer number of backtracks is better. The importance of thorough filtering for the probability variables is clear when the solver is not using the TD heuristic (see Figure 5.7(a)). The distribution of the results on the total number of backtracks shows that while achieving better or equal objective values in the vast majority of the cases when using fractional knapsack filtering, the solver does so in less backtracks compared to elementary constraints filtering.

That being said and even though we demonstrated that the theoretical consistency achieved by the fractional knapsack algorithm is better than the one achieved by elementary constraints (with or without implied constraints), some cases might occurs in practice where a thorough filtering is not improving the performance of the solver. This may be due, for instance, to the interaction between a heuristic (e.g., the TD heuristic) and the solving process. Good heuristics do not branch on inconsistent values and thus require less filtering. This is partly what we observed when using the Mtc along with the TD heuristic from Chapter 4, i.e., when comparing the *MTC-OSP-TD-FK* model to the *MTC-OSP-TD-IA-* model. The objective value of the solutions found by both methods within 10 minutes of allowed solving time (or 5,000,000 backtracks) being similar in presence of the TD heuristic, we use the distribution of the total number of backtracks for comparison purposes (see Figure 5.7(b)). Again, in the vast majority of the cases, the solver benefits from the additional filtering provided by the fractional knapsack algorithm to reduce the total number of backtracks.



(a) *MTC-OSP-IA-* against *MTC-OSP-FK*: comparison of the filtering in number of backtracks



(b) *MTC-OSP-TD-IA-* against *MTC-OSP-TD-FK*: comparison of the filtering in number of backtracks

Figure 5.7: Comparison of the filtering triggered by the elementary constraints to the one triggered by the fractional knapsack filtering algorithm for the Mtcs with and without the TD heuristic; each dot is a comparison of the incumbent solutions found by the compared methods on a single OSP problem instance. The complexity of the instance (in terms of allowed time steps T) is represented by a gradient from black to blue. Fewer backtracks is better.

5.4 Further Thoughts on the Filtering of Markov Transitions

The development of global constraints related to Markov processes has just recently begun (see Section 1.4.2 of Chapter 1). Given the broad interest in Markov processes and the extensive literature on the subject, it seems clear to us that major improvements are still to come for this type of constraints. The Markov transition constraint (Mtc) makes no exception. We saw that linear programming is at least as hard as the problem of filtering the probability variables of a single Mtc in the general case. That is, it always enforces bounds consistency. We also saw that, in some specific cases, the problem of filtering an Mtc is tractable. One possible avenue to improve the filtering of an Mtc is to explore the theory of Markov chains to determine if there exist other cases where filtering the domains of the variables in the scope of the constraint would be easier than solving linear programs. A filtering algorithm could exploit these tractable cases during filtering. The generalization of the constraint to the case of imprecise Markov Chains (see Section 1.4.3 of Chapter 1) is another promising avenue for further research that would enhance CP expressiveness. Cases where the transition matrix $\mathbf{M}$ is uncertain are of interest in practice. It could be useful, for instance, to model the searches in an OSP. The probability mass that transits to the removed state at a given time step is usually uncertain until a search takes place at this time step.

Conclusion

The contributions of this thesis are in two major fields of research linked to path planning: in coverage (a CPP problem generalized to imperfect extended detection) and in detection search-related path planning (the OSP problem). The two problems we dealt with come from two different communities. The CPP is a challenge in mobile robotics. The OSP is a classic from search theory. We presented in Chapter 2 a review of both problems. We can see, from a study of both the CPPIED problem and the OSP problem, that these formalisms are complementary from a modeling point of view. A particularity of the CPPIED is that it really comes close to the OSP from search theory. First, it uses the concept of detection model. Second, coverage paths can be seen as search patterns. It has been discussed, in Chapter 2 that search theory problems usually deals with discretized environments with large sub-areas (regions of several square nautical miles in the case of search operations at sea). In each of these large regions, an assumption on the sensor's search pattern is provided by the decision maker in order to completely define the detection model of the searcher (search unit). Search operations can easily be seen in terms of macro and micro planning. A macro planner would attribute the search effort to large regions and then, a micro planner would find a coverage pattern over the large region in order to guarantee that the required probability of detection is achieved. This is only one of the possible interactions between these two problems. From a broader decision making perspective, the two problems can also be independently seen as two tools for search, surveillance, and detection operations. Whenever the prior on the whereabouts of the search object are unknown, the practical problem at hand is suitable to be a CPP variant. Whenever the prior is known, the practical problem is suitable to be an OSP. A further interesting distinction between coverage and search is in the considered goal (objective) and the available resources (constraints). We can see that OSPs have a tendency to maximize the probability of finding the object (the goal) under constrained resources (e.g., time) whereas CPPs have a tendency to minimize expenses (e.g., time required to complete the operation) under a coverage constraint, e.g., a minimal required probability of detecting a search object. These comparisons are possible when adopting the point of view of searches.

Our specific contributions are the following. We presented, in Chapter 3, a generalized CPP with imperfect extended detection (CPPIED) where the robot (or agent) is allowed to survey cells sideways from a distance. The sensor is modeled by using conditional detection probabilities for the scans [Gage, 1993]. This concept is similar to that of probability of detection (or detection model) found in search theory for the optimal search path (OSP) problem [Stone, 2004]. Since that perfect coverage in presence of imperfect sensor is not possible, the goal in the CPPIED is to guarantee that a minimal required coverage is achieved by the robot. Following [Drabovich, 2008], we consider that an efficient path is of minimal length while also minimizing the number of turns as our tie breaking criterion. We positioned our research with respect to applications in coverage for underwater minesweeping operations. For the discretized seabed maps involved in these coverage problems are made of more than 21 thousands cells and thus involve a huge combinatorial space, we chose to develop a heuristic algorithm. It turned out that our heuristic, we called the dynamic programming sweeper (or simply DpSweeper) outperformed the existing technique from the literature [Drabovich, 2008] by providing short coverage path within a minute or so even on the hardest problem instances in our benchmark. Since a cell by cell (move by move) path planning algorithm is likely to be inefficient on very large maps, our algorithm uses a decomposition of the problem in two phases. During the first phase of the algorithm, we greedily construct a partial path made of disconnected segments. A robot traversing all these disconnected segments is guaranteed to achieve the minimal required coverage. The first subproblem to solve when creating the required set of disconnected segments is to choose, for a given row or column of the seabed grid, a plausible segment of optimal length. This problem, when solved independently for each row (column), is the maximal subarray problem which is solvable in polynomial time using Kadane's algorithm [Bentley, 1984], an example of dynamic programming. The second subproblem to solve is to choose a set made of disconnected segments to add to the partial path. Distant sensor scans render the coverage problem much more difficult since detections may easily overlap. Overlapping detections are required to solve many CPPIED problem instances due to the imperfectness of the sensor. We chose to select a set of disconnected segments with non overlapping detection at a time to add it to the disconnected partial path. While enabling the use of dynamic programming (segments with non overlapping detections do not interfere with each other), the algorithm has a tendency to produce parallel segments which is an intuitive form of coverage pattern. Sets of disconnected segments with non overlapping detection are added to the partial path until the point where the required coverage is achieved in each cell of the seabed map. The second phase of the problem consists in reconnecting the segments of the partial path. This is done by using a traveling salesman problem (TSP) reduction we developed. We used, to solve the TSP problem instance of the second phase, the Concorde TSP solver [Applegate et al., 2011], a solver capable of closing many instances (some of which having more than 2 thousands nodes) of TSPLIB [Reinelt, 1991] within a few minutes. We also discussed the applicability of the CPPIED problem and of our heuristic algorithm in a context more general than the one of underwater minesweeping operations and concluded that the use of both in obstacle-free environments is straightforward. The case of environments with obstacles would require adapting the algorithm and the model so that the generated segments and the TSP reduction consider the obstacles. This would be a generalization of the problem.

In Chapters 4 and 5, we dealt with a different, but related, problem from search theory: the OSP problem. We presented, in Chapter 4, a novel CP model to solve the OSP problem. We first improved on using the intuitive objective function found in the problem definition. Our novel objective function, involving a single modification of the operators used to compute the probability of success of a search, leads to a stronger filtering of the probability variables used in the model to represent the whereabouts of a search object during the operation. While being useful, this proved to be insufficient to drastically improve the CP solver performance. We thus developed a novel heuristic based on search games from graph theory. We called this heuristic the total detection (TD) heuristic in that it computes the total probability of detecting the search object in the remaining time. Our experiments first showed that the heuristic, when used to select the next destination of the searcher in our CP model of the OSP, improves on the objective value found by the solver. Furthermore, it proved to improve on the performance of the solver compared to impact-based search [Refalo, 2004], an efficient general purpose heuristic in CP. We presented the heuristic in a CP context to select, at each time step, the next destination of the searcher. However, we consider it as a general heuristic for the OSP problem that is applicable to solving techniques such as mixed-integer programming or local search.

In Chapter 5, we went further on with the study of the OSP in CP. The main contribution of this chapter is, nonetheless, general. We developed the Markov transition constraint (Mtc). The Mtc, a novel global constraint, is a useful tool to filter CP models with probability variables involved in a Markov chain. Markov chains are a widely used modeling tool. We provided, by introducing the Mtc, an accessible way to model Markov chains in CP. We proved, both empirically and theoretically, that interval arithmetic is insufficient to enforce bounds consistency of a single Mtc. It turns out that interval arithmetic is the only algorithm available to a solver when we decompose the Mtc into individual arithmetic constraints. We developed, for the purpose of enforcing bounds consistency on the probability variables of an Mtc, a filtering algorithm based on linear programming. We also provided, as an in-between solution improving on the filtering of interval arithmetic, a fractional knapsack filtering heuristic. The fractional knapsack filtering algorithm for the Mtc enforces bounds consistency in the case of forward reasoning, the usual way of solving a Markov chain. We applied the Mtc to a CP model of the OSP. We showed that the novel global constraint improved on the filtering performance of the solver on that problem.

That being said, even though we adopted the point of view of search and coverage operations thorough this thesis, the developed tools are meaningful in other contexts as well. Our formalisms deal with general notions found in Markov chains. There are, in these models, decisions taken at different points in time that influence the chain. The proposed formalisms and the solutions to such issues, while being highly applicable to search and coverage problems, are far from being single purpose solutions. It can be seen, from the myriad of applications related to the matter found in the literature (see Chapter 2) that coverage is of capital importance. Even the simplest coverage problems in terms of definition (such as the TSP) have applications ranging from art to circuit boards soldering [Cook, 2012]. We can also think of the applicability of search theory to a myriad of cases including the ones that are farther from being purely a search operation such as medical diagnosis [Iida, 1992]. Not surprisingly, the idea of allocating “search effort” is even found in combinatorial optimization [Tsang et al., 1999]. Is it not exactly what the solver does when searching for solutions? A solver’s, or more generally an algorithm’s, sole aim is often to efficiently cover or search a decision space, a concept notably exploited by Russell and Norvig [2013] in their book on artificial intelligence. Outside the combinatorial world of computers, artificial intelligence and problems solving, it remains that coverage and search problems have important humanitarian applications from search and rescue to minesweeping. All in all, clever algorithms and solutions should ultimately aim at supporting such important decisions from various domains which remains an on-going and day-to-day challenge in research.

Appendix A

List of Symbols

COS	A *cos* constraint programming variable	73–75, 102
cos	The *cumulative overall probability of success* of a search plan has different flavors depending on the search theory problem to solve. For an OSP problem, the *cos* up to a time t is defined as $cos_t = \sum_{t' \leq t} \sum_{r \in \mathcal{V}(G_A)} pos_{t'}(r)$.	42, 45–49, 52, 71, 73, 74, 101
dom	The *domain* of a variable is the set of plausible values to which that variable can be instantiated.	11–18, 20, 28, 75, 81, 87, 88, 90, 91, 93, 102, 112, 114
$\mathcal{FSET}$	Given a combinatorial optimization problem instance, the *feasible solution set* is the set of all solutions that satisfy the constraints of the problem instance, i.e., the set of all candidate solutions.	10, 25
$\mathcal{H}^*, \mathcal{V}^*$	In the DpSweeper algorithm context, $\mathcal{H}^*$ (resp. $\mathcal{V}^*$) is the subset of segments that maximizes the sum of the horizontal (resp. vertical) gains over the rows (resp. columns) subject to a non-overlapping detection constraint.	59–63
$\mathbf{M}$	A Markovian transition matrix	26–28, 48, 73–75, 77, 80, 86, 88–95, 99–103, 106
$\mathcal{N}$	The state space of a Markov chain	26, 28, 87–93, 95

$\mathbf{O}$	The seabed map matrix in a CPPIED problem	55–58, 60, 65
POC^{search}	A *poc* constraint programming variable that takes searches into account.	101, 102
POC	A *poc* constraint programming variable	73, 74, 76, 81, 101, 102
poc	The *probability of containment* distribution that carries the knowledge on the whereabouts of a search object. In an OSP problem, the evolution of the whereabouts is subject to the motion of the search object and to the searches. We have that $poc_t(r) = \sum_{s \in \mathcal{V}(G_A)} \mathbf{M}_{sr} \left[poc_{t-1}(s) - pos_{t-1}(s)\right]$.	42, 46–50, 73, 75, 76, 100–102
pod	The *probability of detection* definition varies with respect to the considered search theory problem. In an OSP, it is the probability of detecting the object when both the searcher and the object are co-located.	44–48, 50, 73–77, 80, 81, 84, 101–103
POS	A *pos* constraint programming variable	73–76, 101, 102
pos	In an OSP problem, the (local) *probability of success* in a region (vertex) r at a time t is defined as $pos_t(r) = poc_t(r) \times pod(r)$.	46, 48, 50, 75, 101
p^{scan}	The *conditional detection probability* of a sensor scan is the detection model of the searcher (robot or agent) in a CPPIED problem. It is defined by a $\lvert\mathcal{T}\rvert \times r^{\max}$ matrix with the seabed types on rows and the lateral distances in number of cells on columns.	56–58, 60, 65
$r^{\max}$	The *maximal lateral range* of a sidescan sonar in number of cells in a CPPIED problem.	55–57, 59–62, 65
$\mathcal{S}$	In the DpSweeper algorithm context, $\mathcal{S}$ is the set of disconnected segments used to construct the path of the searcher (robot, agent) so that the required coverage $\mathbf{D}$ is achieved.	59–64
$\mathcal{T}$	The set of seabed types in the CPPIED problem	55–57, 60, 62
$\overline{X}$	The *upper bound* of the domain of a constraint programming variable X is $\overline{X} = \max_{x \in \text{dom}(X)} x$.	12, 16, 17, 87, 88, 90–92, 94
$\underline{X}$	The *lower bound* of the domain of a constraint programming variable X is $\underline{X} = \min_{x \in \text{dom}(X)} x$.	12, 16, 17, 75, 87, 88, 90–94

Appendix B

Complement to Domain Filtering Examples in Constraint Programming

We presented, in Section 1.2.1 of Chapter 1, two examples of search trees for a CP model of the HCP problem over the graph G reproduced here on Figure B.1. In the first example, Example 1.2.3, we assume a solver that branches in a depth-first fashion as it explores the search space of the tour variables. The solver backtracks 15 times when using this strategy. In the second example, Example 1.2.4, we assume that the solver implements filtering algorithms for the constraints used in the model. The solver does not backtrack when using this strategy on this problem instance which is an improvement over using a depth-first search with backtracking and without filtering. Table B.1 completes Example 1.2.4 by providing details on the solver's actions. Values are selected in the lexicographic order they appear in the domain of the tour variables.

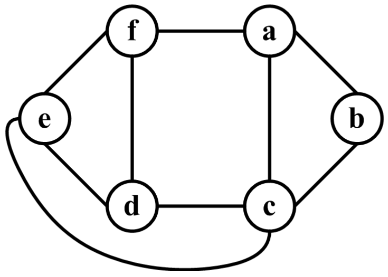


Figure B.1: The HCP instance of Examples 1.2.3 and 1.2.4 of Section 1.2.1 of Chapter 1

Table B.1: Domains of the HCP instance of Example 1.2.4 from Section 1.2.1 of Chapter 1 when using a depth-first search with filtering

Domains						**Solver's actions**
dom($TOUR_1$)	dom($TOUR_2$)	dom($TOUR_3$)	dom($TOUR_4$)	dom($TOUR_5$)	dom($TOUR_6$)	
Depth 0						
$\{a,b,c,d,e,f\}$	$\{a,b,c,d,e,f\}$	$\{a,b,c,d,e,f\}$	$\{a,b,c,d,e,f\}$	$\{a,b,c,d,e,f\}$	$\{a,b,c,d,e,f\}$	Opens the nodes for $TOUR_1$
$\{a\}$	$\{a,b,c,d,e,f\}$	$\{a,b,c,d,e,f\}$	$\{a,b,c,d,e,f\}$	$\{a,b,c,d,e,f\}$	$\{a,b,c,d,e,f\}$	Branches on $TOUR_1 = a$
$\{a\}$	$\left\{\cancel{a},b,c,\cancel{d},\cancel{e},f\right\}$	$\left\{\cancel{a},b,c,d,e,f\right\}$	$\left\{\cancel{a},b,c,d,e,f\right\}$	$\left\{\cancel{a},b,c,d,e,f\right\}$	$\left\{\cancel{a},b,c,\cancel{d},\cancel{e},f\right\}$	AllDifferent($TOUR_1,\ldots,TOUR_6$) filters a from dom($TOUR_i$) ($\forall i$) ($TOUR_1, TOUR_2$) $\in \mathcal{E}(G)$ filters d, and e from dom($TOUR_2$) ($TOUR_6, TOUR_1$) $\in \mathcal{E}(G)$ filters d, and e from dom($TOUR_6$)
Depth 1						
$\{a\}$	$\{b,c,f\}$	$\{b,c,d,e,f\}$	$\{b,c,d,e,f\}$	$\{b,c,d,e,f\}$	$\{b,c,f\}$	Opens the nodes for $TOUR_2$
$\{a\}$	$\{b\}$	$\{b,c,d,e,f\}$	$\{b,c,d,e,f\}$	$\{b,c,d,e,f\}$	$\{b,c,f\}$	Branches on $TOUR_2 = b$
$\{a\}$	$\{b\}$	$\left\{\cancel{b},c,\cancel{d},\cancel{e},\cancel{f}\right\}$	$\left\{\cancel{b},\cancel{c},d,e,\cancel{f}\right\}$	$\left\{\cancel{b},\cancel{c},d,e,\cancel{f}\right\}$	$\left\{\cancel{b},\cancel{c},f\right\}$	AllDifferent($TOUR_1,\ldots,TOUR_6$) filters b from dom($TOUR_i$) ($\forall i$) ($TOUR_2, TOUR_3$) $\in \mathcal{E}(G)$ filters d, e, and f from dom($TOUR_3$) AllDifferent($TOUR_1,\ldots,TOUR_6$) filters c from dom($TOUR_i$) ($\forall i$) ($TOUR_3, TOUR_4$) $\in \mathcal{E}(G)$ filters f from dom($TOUR_4$) AllDifferent($TOUR_1,\ldots,TOUR_6$) filters f from dom($TOUR_i$) ($\forall i$)
Depth 2						
$\{a\}$	$\{b\}$	$\{c\}$	$\{d,e\}$	$\{d,e\}$	$\{f\}$	Opens the nodes for $TOUR_3$ Branches on $TOUR_3 = c$
Depth 3						
$\{a\}$	$\{b\}$	$\{c\}$	$\{d,e\}$	$\{d,e\}$	$\{f\}$	Opens the nodes for $TOUR_4$
$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{d,e\}$	$\{f\}$	Branches on $TOUR_4 = d$
$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\left\{\cancel{d},e\right\}$	$\{f\}$	AllDifferent($TOUR_1,\ldots,TOUR_6$) filters d from dom($TOUR_i$) ($\forall i$)
Depth 4						
$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	Opens the nodes for $TOUR_5$ Branches on $TOUR_5 = e$
Depth 5						
$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	Opens the nodes for $TOUR_6$ Branches on $TOUR_6 = f$

Bibliography

I. Abi-Zeid and J. R. Frost. SARPlan: A decision support system for canadian search and rescue operations. *European Journal of Operational Research*, 162(3):630–653, 2005.

I. Abi-Zeid, M. Morin, and T. T. Nguyen. Vers une planification multicritère dans le cadre de missions de recherche et sauvetage terrestres. In *73rd Meeting of the European Working Group in Multiple Criteria Decision Aid (EWG-MCDA'11)*, 2011a.

I. Abi-Zeid, O. Nilo, and L. Lamontagne. Resource allocation algorithms for planning search and rescue operations. *Information Systems and Operational Research*, 49(1):15–30, 2011b.

E. Acar, Y. Zhang, H. Choset, M. Schervish, A. G. Costa, R. Melamud, D. Lean, and A. Graveline. Path planning for robotic demining and development of a test platform. In *Proceedings of the 8th International Conference on Field and Service Robotics (FSR'01)*, pages 161–168, 2001.

D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde TSP solver. http://www.tsp.gatech.edu/concorde, 2011. Accessed: 2013/03.

J. Aulinas, Y. R. Petillot, J. Salvi, and X. Lladó. The SLAM problem: A survey. In *Proceedings of the 11th Congress of the ACIA (CCIA'08)*, pages 363–371, 2008.

F. Aurenhammer. Voronoï diagrams–A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

G. Barbieri, F. Pachet, P. Roy, and M. Degli Esposti. Markov constraints for generating lyrics with style. In *Proceedings of 20th biennial European Conference on Artificial Intelligence (ECAI'12)*, 2012.

H. Bast and S. Hert. The area partitioning problem. In *Proceedings of the 12th Canadian Conference on Computational Geometry (CCCG'00)*, 2000.

N. Beldiceanu and S. Demassey. Global constraint catalog. `http://sofdem.github.io/gccat/`, 2014. Accessed: 2014-11.

R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, 1962.

F. Benhamou and L. Grandvilliers. Continuous and interval constraints. In F. Rossi, P. V. Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, pages 571–603. Elsevier, 2006.

S. Benkoski, J. R. Weisinger, and M. G. Monticino. A survey of the search theory literature. *Naval Research Logistics*, 38(4):469–494, 1991.

J. Bentley. Programming pearls: Algorithm design techniques. *Communications of the ACM*, 27(9):865–873, 1984.

J. Berger and N. Lo. An innovative multi-agent search-and-rescue path planning approach. *Computers & Operations Research*, 53:24–31, 2015.

J. Berger, A. Boukhtouta, A. Benmoussa, and O. Kettani. A new mixed-integer linear programming model for rescue path planning in uncertain adversarial environment. *Computers & Operations Research*, 39(12):3420–3430, 2012.

J. Berger, N. Lo, and M. Noel. Exact solution for search-and-rescue path planning. *International Journal of Computer and Communication Engineering*, 2(3):266–271, 2013.

O. Berman, E. Ianovsky, and D. Krass. Optimal search path for service in the presence of disruptions. *Computers & Operations Research*, 38(11):1562–1571, 2011.

C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Filtering algorithms for the NValue constraint. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 79–93, 2005.

C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Filtering algorithms for the NValue constraint. *Constraints*, 11(4):271–293, 2006.

H. Blane and D. den Hertog. On Markov chains with uncertain data. *Tilburg: Operations Research (CentER Dicussion Paper)*, 2008-50:20 p, 2008.

Ø. Breivik, A. A. Allen, C. Maisondieu, and M. Olagnon. Advances in search and rescue at sea. *Ocean Dynamics*, 63(1):83–88, 2012.

K. M. Brown and I. Miguel. Uncertainty and change. In F. Rossi, P. V. Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, pages 731–760. Elsevier, 2006.

S. S. Brown. Optimal search for a moving target in discrete time and space. *Operations Research*, 28(6):1275–1289, 1980.

G. Cannata and A. Sgorbissa. A minimalist algorithm for multirobot continuous coverage. *IEEE Transactions on Robotics*, 27(2):297–312, 2011.

Z. L. Cao, Y. Huang, and E. L. Hall. Region filling operations with random obstacle avoidance for mobile robots. *Journal of Robotic Systems*, 5(2):87–102, 1988.

L. Carlone and D. Lyons. Uncertainty-constrained robot exploration: A mixed-integer linear programming approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'14)*, pages 1140–1147, 2014.

S. Carlsson, J. Håkan, and B. J. Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete & Computational Geometry*, 22(3):377–402, 1999.

R. Chandramouli. Web search steganalysis: Some challenges and approaches. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS'04)*, pages 576–579, 2004.

W.-p. Chin and S. Ntafos. Optimum watchman route. *Information Processing Letters*, 28(1): 39–44, 1988.

H. Choset. Coverage for robotics. *Annals of Mathematics and Artificial Intelligence*, 31(1–4): 113–126, 2001.

T. H. Chung, G. A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, 31(4):299–316, 2011.

N. E. Clarke and G. MacGillivray. Characterizations of k-copwin graphs. *Discrete Mathematics*, 312(8):1421–1425, 2012.

A. Commons. Commons math: The Apache commons mathematics library. http://commons.apache.org/proper/commons-math/, 2010. Accessed: 2013-10.

W. Cook. *In pursuit of the traveling salesman: Mathematics at the limits of computation.* Princeton University Press, 2012.

F. P. A. Coolen, M. C. M. Troffaes, and T. Augustin. Imprecise probability. In M. Lovric, editor, *International Encyclopedia of Statistical Science*, pages 645–648. Springer, 2011.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* The MIT Press, 3rd edition, 2009.

D. R. Cox and H. D. Miller. *The theory of stochastic processes.* Chapman and Hall Ltd, 1972.

G. B. Dantzig. Programming of interdependent activities, II, Mathematical model. *Econometrica: Journal of the Econometric Society*, 17(3/4):200–211, 1949.

M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry, Algorithms and Applications*. Springer-Verlag, Berlin, 3rd edition, 2008.

L. M. de Campos, J. F. Huete, and S. Moral. Probability intervals: A tool for uncertain reasoning. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2(2):167–196, 1994.

L. M. de Campos, J. F. Huete, and S. Moral. Uncertainty management using probability intervals. In *Advances in Intelligent Computing*, pages 190–199. Springer, 1995.

R. N. de Carvalho, H. A. Vidal, P. Vieira, and M. I. Ribeiro. Complete coverage path planning and guidance for cleaning robots. In *Proceedings of IEEE International Symposium on Industrial Electronics (ISIE'97)*, pages 677–682, 1997.

G. de Cooman, F. Hermans, and E. Quaeghebeur. Imprecise Markov chains and their limit behavior. *Probability in the Engineering and Informational Sciences*, 23(4):597–635, 2009.

L. De Floriani and P. Magillo. Algorithms for visibility computation on terrains: A survey. *Environment and Planning B*, 30(5):709–728, 2003.

R. F. Dell, J. N. Eagle, G. Henrique, A. Martins, and A. G. Santos. Using multiple searchers in constrained-path, moving-target search problems. *Naval Research Logistics*, 43(4):463–480, 1996.

J. H. Discenza. *Optimal Search with Multiple Rectangular Search Areas*. PhD thesis, Graduate School of Business Administration, New York University, New York, 1979.

M. Dorigo and C. Blum. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2):243–278, 2005.

M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(1): 29–41, 1996.

M. Drabovich. Automated mission trajectory planning for mine countermeasures operations. Master's thesis, Heriot Watt University, Edinburgh, 2008.

J. N. Eagle. The optimal search for a moving target when the search path is constrained. *Operations Research*, 32(5):1107–1115, 1984.

J. N. Eagle and J. R. Yee. An optimal branch-and-bound procedure for the constrained path, moving target search problem. *Naval Research Logistics*, 38(1):110–114, 1990.

J. W. Eaton. GNU Octave. `http://www.gnu.org/software/octave/`, 2013. Accessed: 2013-10.

A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22 (6):46–57, 1989.

C. Fang and S. Anstee. Coverage path planning for harbour seabed surveys using an autonomous underwater vehicle. In *Proceedings of OCEANS 2010 MTS/IEEE Conference (OCEANS'10)*, pages 1–8, 2010.

R. J. Faudree, R. J. Gould, M. S. Jacobson, and D. B. West. Minimum degree and dominating paths. submitted to Journal of Graph Theory, 2014.

F. V. Foming and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.

J. R. Frost. Principles of search theory, part I: Detection. *Response*, 2(17):1–7, 1999a.

J. R. Frost. Principles of search theory, part II: Effort, coverage and pod. *Response*, 2(17): 8–15, 1999b.

J. R. Frost. Principles of search theory, part III: Probability density distributions. *Response*, 3(17):1–10, 1999c.

J. R. Frost. Principles of search theory, part IV: Optimal effort allocation. *Response*, 3(17): 11–23, 1999d.

A. Fruitet. Planification de chemins de couverture avec capteurs imparfaits en environnements hétérogènes. Rapport de stage M1 ISMAG, Université Toulouse le Mirail. Toulouse, 2013.

Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*, 31(1):77–98, 2001.

D. W. Gage. Randomized search strategies with imperfect sensors. In *Proceedings of SPIE Mobile Robots VIII*, pages 270–279, 1993.

D. W. Gage. Many-robot MCM search systems. In A. Bottoms, J. Eagle, and H. Bayless, editors, *Proceedings of Autonomous Vehicles in Mine Countermeasures Symposium*, 1995.

E. Galceran. Towards coverage path planning for autonomous underwater vehicles. Master's thesis, Universitat de Girona, Girona, 2011.

E. Galceran and M. Carreras. Coverage path planning for marine habitat mapping. In *Proceedings of OCEANS 2012 MTS/IEEE Conference (OCEANS'12)*, pages 1–8, 2012.

M. R. Garey and D. S. Johnson. *Computers and intractability.* Freeman, 1979.

C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *Journal of Intelligent and Robotic Systems*, 57 (1-4):65–100, 2010.

M. F. Goodchild and J. Lee. Coverage problems and visibility regions on topographic surfaces. *Annals of Operations Research*, 18(1):175–186, 1989.

R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

A. Guitouni and H. Masri. An orienteering model for the search and rescue problem. *Computational Management Science*, 11(4):459–473, 2014.

G. Hahn and G. MacGillivray. A note on k-cop, l-robber games on graphs. *Discrete Mathematics*, 306(19):2492–2497, 2006.

J. D. Hamilton. A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica: Journal of the Econometric Society*, 57(2):357–384, 1989.

M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial & Applied Mathematics*, 10(1):196–210, 1962.

H. H. Hoos and T. Stützle. *Stochastic local search: Foundations & applications.* Morgan Kaufmann / Elsevier, 2004.

H. H. Hoos and E. Tsang. Local search methods. In F. Rossi, P. V. Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, pages 135–167. Elsevier, 2006.

W. H. Huang. The minimal sum of altitudes decomposition for coverage algorithms. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'01)*, pages 27–32, 2000.

K. Iagnemma and S. Dubowsky. *Mobile robots in rough terrain: Estimation, motion planning, and control with application to planetary rovers.* Springer Tracts in Advanced Robotics. Springer, 2004.

IBM. IBM CPLEX Optimizer. `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`, 2013. Accessed: 2013-10.

K. Iida. *Studies on the optimal search plan.* Springer, New York, 1992.

A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

P. A. Jimenez, B. Shirinzadeh, A. Nicholson, and G. Alici. Optimal area covering using genetic algorithms. In *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM'07)*, pages 1–5, 2007.

D. Joho, M. Senk, and W. Burgard. Learning search heuristics for finding objects in structured environments. *Robotics and Autonomous Systems*, 59(5):319–328, 2011.

H. Karami, S. M. Sheikholeslami, A. Khodkar, and D. B. West. Connected domination number of a graph and its complement. *Graphs and Combinatorics*, 28(1):123–131, 2012-01.

I. Katriel, M. Sellmann, E. Upfal, and P. Van Hentenryck. Propagating knapsack constraints in sublinear time. In *Proceedings of the 22nd national conference on artificial intelligence (AAAI'07)*, pages 231–236, 2007.

A. Kehagias and P. Prałat. Some remarks on cops and drunk robbers. *Theoretical Computer Science*, 463:133–147, 2012.

A. Kehagias, D. Mitsche, and P. Prałat. Cops and invisible robbers: The cost of drunkenness. *Theoretical Computer Science*, 481:100–120, 2013.

Y.-H. Kim, S. Rana, and S. Wise. Exploring multiple viewshed analysis using terrain features and optimisation techniques. *Computers & Geosciences*, 30(9):1019–1032, 2004.

R. Koester, D. C. Cooper, J. R. Frost, and R. Q. Robe. Sweep width estimation for ground search and rescue. Technical report, United States Coast Guard Operations, Washington, 2004.

A. Kolling and S. Carpin. The graph-clear problem: Definition, theoretical properties and its connections to multirobot aided surveillance. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'07)*, pages 1003–1008, 2007.

A. Kolling and S. Carpin. Probabilistic graph-clear. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'09)*, pages 3508–3514, 2009a.

A. Kolling and S. Carpin. Surveillance strategies for target detection with sweep lines. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'09)*, pages 5821–5827, 2009b.

A. Kolling and S. Carpin. Solving pursuit-evasion problems with graph-clear: An overview. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'10). Workshop: Search and Pursuit/Evasion in the Physical World: Efficiency, Scalability, and Guarantees*, pages 27–32, 2010.

N. Komarov and P. Winkler. Capturing the drunk robber on a graph. *arXiv preprint arXiv:1305.4559*, 2013.

B. O. Koopman. The theory of search. I. Kinematic bases. *Operations Research*, 4(3):324–346, 1956a.

B. O. Koopman. The theory of search. II. Target detection. *Operations Research*, 4(5): 503–531, 1956b.

B. O. Koopman. The theory of search. III. The optimum distribution of searching effort. *Operations Research*, 5(5):613–626, 1957.

B. O. Koopman. *Search and Screening: General Principles with Historical Applications*. Pergamon Press, New York, 1980.

E. Koutsoupias, C. H. Papadimitriou, and M. Sideri. On the optimal bisection of a polygon. *ORSA Journal on Computing*, 4(4):435–438, 1992.

E. Kranakis, D. Krizanc, and S. Rajsbaum. Computing with mobile agents in distributed networks. In S. Rajasedaran and J. Reif, editors, *Handbook of Parallel Computing: Models, Algorithms, and Applications*, pages 8.1–8.20. CRC Press, 2007.

M. Kulich, L. Přeučil, and J. J. M. Bront. Single robot search for a stationary object in an unknown environment. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'14)*, pages 5830–5835, 2014.

J.-M. Labat and J.-C. Pomerol. Are branch and bound and A* algorithms identical? *Journal of Heuristics*, 9(2):131–143, 2003.

F. Laburthe and N. Jussien. *Choco Solver Documentation*, 2012. http://www.emn.fr/z-info/choco-solver/.

H. Lau. *Optimal Ssearch in Structured Environments*. PhD thesis, The University of Technology, Sydney, 2007.

H. Lau, S. Huang, and G. Dissanayake. Probabilistic search for a moving target in an indoor environment. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'06)*, pages 3393–3398, 2006.

H. Lau, S. Huang, and G. Dissanayake. Discounted mean bound for the optimal searcher path problem with non-uniform travel times. *European journal of operational research*, 190 (2):383–397, 2008.

S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Iowa State University, Iowa, 1998.

S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

S. M. LaValle and J. J. K. Jr. Rapidly-exploring random tress: Progress and prospects. *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2001.

X. R. Li and V. P. Jilkov. A Survey of Maneuvering Target Tracking. Part I. Dynamic Models. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4):1333–1364, 2003.

S.-H. Lim and H.-C. Bang. Waypoint planning algorithm using cost functions for surveillance. *International Journal of Aeronautical and Space Sciences*, 11(2):136–144, 2010.

G. F. Luger. *Artificial intelligence: Structures and strategies for complex problem solving*. Addison-Wesley, 2005.

A. Macwan. *A Multi-Robot Coordination Methodology for Wilderness Search and Rescue*. PhD thesis, University of Toronto, Toronto, 2013.

G. Mahalingam. *Connected domination in graphs*. PhD thesis, University of South Florida, Tampa, 2005.

A. Makhorin. GLPK (GNU Linear Programming Kit). http://www.gnu.org/software/glpk/, 2012. Accessed: 2013-10.

R. Mannadiar and I. Rekleitis. Optimal coverage of a known arbitrary environment. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'10)*, pages 5525–5530, 2010.

H. Marcoux. Jeux de poursuite policier-voleur sur un graphe. Master's thesis, Université Laval, Québec, 2014.

G. H. Martins. A new branch-and-bound procedure for computing optimal search paths. Master's thesis, Naval Postgraduate School, 1993.

Mathworks. MATLAB - The Language of Technical Computing. http://www.mathworks.com/products/matlab/, 2010. Accessed: 2013-10.

N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *Journal of the ACM (JACM)*, 35(1):18–44, 1988.

K. Miettinen. Introduction to multiobjective optimization: Noninteractive approaches. In *Multiobjective Optimization*, pages 1–26. Springer, 2008.

K. Miettinen, F. Ruiz, and A. P. Wierzbicki. Introduction to multiobjective optimization: Interactive approaches. In *Multiobjective Optimization*, pages 27–57. Springer, 2008.

R. E. Moore. *Interval analysis*. Prentice Hall, 1966.

M. Morin. Multi-criteria path planning with terrain visibility constraints: The optimal searcher path problem with visibility. Master's thesis, Université Laval, Québec, 2010.

M. Morin and C.-G. Quimper. The Markov transition constraint. In *Integration of AI and OR Techniques in Constraint Programming*, pages 405–421, 2014.

M. Morin, L. Lamontagne, I. Abi-Zeid, P. Lang, and P. Maupin. The optimal searcher path problem with a visibility criterion in discrete time and space. In *Proceedings of the 12th International Conference on Information Fusion*, pages 2217–2224, 2009.

M. Morin, L. Lamontagne, I. Abi-Zeid, and P. Maupin. The ant search algorithm: An ant colony optimization algorithm for the optimal searcher path problem with visibility. In *Advances in Artificial Intelligence: 23rd Canadian Conference on Artificial Intelligence*, pages 196–207, 2010.

M. Morin, A. P. Papillon, F. Laviolette, I. Abi-Zeid, and C.-G. Quimper. Constraint programming for path planning with uncertainty: Solving the optimal search path problem. In *Principles and Practice of Constraint Programming*, pages 988–1003, 2012.

M. Morin, I. Abi-Zeid, T. T. Nguyen, L. Lamontagne, and P. Maupin. Search and surveillance in emergency situations – A GIS based approach to construct optimal visibility graphs. In *Proceedings of the 10th International Conference on Information Systems for Crisis Response and Management (ISCRAM'13)*, pages 452–456, 2013a.

M. Morin, I. Abi-Zeid, Y. R. Petillot, and C.-G. Quimper. A hybrid algorithm for coverage path planning with imperfect sensors. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'13)*, 2013b.

A. T. Murray, K. Kim, J. W. Davis, R. Machiraju, and R. Parent. Coverage optimization to support security monitoring. *Computers, Environment and Urban Systems*, 31(2):133–147, 2007.

R. Netsch. The USCG search and rescue optimal planning system (SAROPS) via the commercial / joint mapping tool kit (C/JMTK). Technical report, Department of Homeland Security, U.S. Coast Guard, 2004.

B. Nguyen and D. Hopkin. Modeling autonomous underwater vehicle (AUV) operations in mine hunting. In *Proceedings of OCEANS 2005 MTS/IEEE Conference (OCEANS'05)*, pages 533–538, 2005.

J. W. Nicholson and A. J. Healey. The present state of autonomous underwater vehicle (AUV) applications and technologies. *Marine Technology Society Journal*, 42(1):44–51, 2008.

R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43(2-3):235–239, 1983.

S. Ntafos. Watchman routes under limited visibility. *Computational Geometry*, 1(3):149–170, 1992.

T. Oksanen and A. Visala. Coverage path planning algorithms for agricultural field machines. *Journal of Field Robotics*, 26(8):651–668, 2009.

J. O'Madadhain, D. Fisher, T. Nelson, S. White, and Y. B. Boey. JUNG: Java universal network/graph framework. http://jung.sourceforge.net, 2010. Accessed: 2015-07.

F. Pachet and P. Roy. Markov constraints: Steerable generation of Markov sequences. *Constraints*, 16(2):148–172, 2011.

F. Pachet, P. Roy, and G. Barbieri. Finite-length Markov processes with constraints. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 635–642, 2011.

L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: Algorithms and complexity.* Dover Publications, 1998.

T. D. Parsons. Pursuit-evasion in a graph. *Theory and applications of graphs*, 642:426–441, 1978.

L. Paull, S. Saeedi, and H. Li. Path planning for autonomous underwater vehicles. In *Marine Robot Autonomy*, pages 177–223. Springer, 2013.

A. Quilliot. Problème de jeux, de point fixe, de connectivité et de représentation sur des graphes, des ensembles ordonnés et des hypergraphes, 1983. Université Paris IV, Thèse d'État.

S. Reed, Y. Petillot, and J. Bell. An automatic approach to the detection and extraction of mine features in sidescan sonar. *IEEE Journal of Oceanic Engineering*, 28(1):90–105, 2003.

P. Refalo. Impact-based search strategies for constraint programming. In *Principles and Practice of Constraint Programming*, pages 557–571, 2004.

J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National conference on Artificial Intelligence (AAAI'94)*, pages 362–367, 1994.

G. Reinelt. TSPLIB–A traveling salesman problem library. *ORSA Journal on Computing*, 3 (4):376–384, 1991.

D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

F. Royer. Étude de la sélection des segments dans un algorithme de planification de couverture avec capteurs imparfaits en environnements hétérogènes. Rapport de stage Polytech Nantes. Nantes, 2014.

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 3rd edition, 2013.

H. M. Salkin. *Integer programming.* Addison-Wesley, 1975.

H. Sato. *Path optimization for single and multiple searchers: Models and algorithms.* PhD thesis, Naval Postgraduate School, Monterey, CA, 2008.

H. Sato and J. O. Royset. Path optimization for the resource-constrained searcher. *Naval Research Logitics*, 57(5):422–440, 2010.

A. Schrijver. On the history of combinatorial optimization (till 1960). *Handbooks in Operations Research and Management Science: Discrete Optimization*, 12:1–68, 2005.

F. Simard, M. Morin, C.-G. Quimper, F. Laviolette, and J. Desharnais. Relaxation of the optimal search path problem with the cop and robber game. In *Doctoral Program of the 20th Conference on Principles and Practice of Constraint Programming (CP 2014)*, 2014.

F. Simard, M. Morin, C.-G. Quimper, F. Laviolette, and J. Desharnais. Bounding an optimal search path with a game of cop and robber on graphs. In *Principles and Practice of Constraint Programming*, pages 403–418, 2015.

D. Škulj. Discrete time Markov chains with interval probabilities. *International Journal of Approximate Reasoning*, 50(8):1314–1329, 2009.

B. M. Smith. Modelling. In F. Rossi, P. V. Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, pages 377–406. Elsevier, 2006.

D. Smith and S. Singh. Approaches to multisensor data fusion in target Tracking: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 18(12):1696–1710, 2006.

J. R. Stack and C. M. Smith. Combining random and data-driven coverage planning for underwater mine detection. In *Proceedings of OCEANS 2003 MTS/IEEE Conference (OCEANS'03)*, pages 2463–2468, 2003.

T. J. Stewart. Search for a moving target when the searcher motion is restricted. *Computers & Operations Research*, 6(3):129–140, 1979.

L. D. Stone. The process of search planning: Current approaches and continuing problems. *Operations Research*, 31(2):207–233, 1983.

L. D. Stone. Search for the SS Central America: Mathematical treasure hunting. *Interfaces*, 22(1):32–54, 1992.

L. D. Stone. *Theory of Optimal Search.* Academic Press, 2004.

E.-G. Talbi. *Metaheuristics: From design to implementation.* John Wiley & Sons, Hoboken, New Jersey, 2009.

S. A. Tarim, S. Manandhar, and T. Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80, 2006.

K. E. Trummel and J. R. Weisinger. The complexity of the optimal searcher path problem. *Operations Research*, 34(2):324–327, 1986.

E. P. Tsang, C. J. Wang, A. Davenport, C. Voudouris, and T. L. Lau. A family of stochastic methods for constraint satisfaction and optimization. In *The First International Conference on the Practical Application of Constraint Technologies and Logic Programming (PACLP'99)*, pages 359–383, 1999.

P. van Beek. Backtracking search algorithms. In F. Rossi, P. V. Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, pages 85–134. Elsevier, 2006.

W.-J. van Hoeve and I. Katriel. Global constraints. In F. Rossi, P. V. Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, pages 169–208. Elsevier, 2006.

G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments: A survey. *Constraints*, 10(3):253–281, 2005.

M. Verkama. Optimal paging - a search theory approach. In *Proceedings of the 5th IEEE International Conference on Universal Personal Communications (ICUPC'96)*, pages 956–960, 1996.

A. R. Washburn. Search for a moving target: The FAB algorithm. *Operations Research*, 31 (4):731–751, 1983.

A. R. Washburn. Branch and bound methods for a search problem. *Naval Research Logistics*, 45(3):243–257, 1998.

K. Weichselberger. The theory of interval-probability as a unifying concept for uncertainty. *International Journal of Approximate Reasoning*, 24(2):149–170, 2000.

M. Weiss-Cohen, I. Sirotin, and E. Rave. Lawn mowing system for known areas. In *Proceedings of the IEEE International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA'08)*, pages 539–544, 2008.

E. W. Weisstein. Sigma-algebra. From MathWorld–A Wolfram web resource. `http://mathworld.wolfram.com/Sigma-Algebra.html`, 2014. Accessed: 2014-12.

Wikipedia. Side-scan sonar. http://en.wikipedia.org/wiki/Side-scan_sonar, 2013. Accessed: 2014-01.

D. P. Williams. On optimal AUV track-spacing for underwater mine detection. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'10)*, pages 4755–4762, 2010.

A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys*, 38 (4):1–45, 2006.

Y. Zhang, M. Schervish, E. U. Acar, and H. Choset. Probabilistic methods for robotic landmine search. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'01)*, pages 1525–1532, 2001.