Multi-Robot Patrolling and Other Multi-Robot Cooperative Tasks: An Algorithmic Approach

Noa Agmon
Computer Science Department

Ph.D. Thesis

Submitted to the Senate of Bar-Ilan University
Ramat Gan, Israel
February 2009

This work was carried out under the supervision of Prof. Sarit Kraus and Prof. Gal A. Kaminka, Computer Science Department, Bar-Ilan University.

Acknowledgments

This thesis summarizes my research that was carried out over the past four years, during which time I had the opportunity to stumble along many sensible, insightful, kind and helpful people who have inspired and supported me throughout my journey.

First and foremost, I would like to thank my advisors Professor Sarit Kraus and Professor Gal Kaminka. Working with them, together and apart, has been an inspiring experience. It is not often in life, and especially in the academic world, that one can find two sharp minds that work together in such a perfect cohesion, yet still leave room for other ideas to blossom. Their total support and tremendous encouragement (both professionally and personally) have made this journey fruitful and enjoyable. I could not have asked for better mentors.

I would also like to thank my team members—from both MAVERICK and the MAS groups—for creating a friendly, productive environment. Special thanks to Yehuda Elmaliach for introducing the patrolling world to me, Efrat Manisterski, for her help in untying a difficult knot in the patrolling work, to Noam Hazon and Asaf Shiloni for the pleasure of working with them and producing nice papers on coverage and ants (respectively), and to Vova Sadov for his effort in the execution of the patrolling game.

To my parents, who have been the best and most enthusiastic advocators of the academic life and thus, along with their total support and encouragement, made the choice of continuing my studies the most trivial decision I had to make. To my sisters, Idit and Tammy, whom I could always count on for their support (and on their perfect hosting, when traveling to conferences overseas), and to my in laws, Edna and Zvika, whom without their help with babysitting services and a lot more, many deadlines would have been impossible to meet.

Finally, and most importantly, I would like to thank my husband Doron, whose love, patience and encouragement have made it possible for me to complete this work. Also to my sons Ailon and Dothan, for their positive, innocent and smart view of life which have helped open my mind to new ideas, and most importantly - made my life considerably more challenging, satisfying and complete.

This research was supported in part by ISF grant #1357/07 and #1685/07.

Abstract

The subject of cooperative multi robot systems has been thoroughly investigated over the past decade. These systems have immediate applicability in a wide variety of tasks, such as military operations and space missions. Building algorithms for multi-robot systems is a complex challenge, for which solutions and methods are brought from many different disciplines. There is significant interest in multi-robot systems also in the discipline of theoretical computer science, mainly by using distributed systems analysis. However, researchers who have worked on multi-robot systems from a distributed algorithms perspective have tended to make unrealistic assumptions about the robots' capabilities, resulting in algorithms impractical for realistic systems. In our work, we were motivated by methods from the theoretical computer science discipline, yet we incorporated realistic assumptions on the robots' capabilities.

In the first and the main part of the work we consider the task of multi-robot patrol in adversarial environments. In this task, a team of robots is required to continuously visit some target area in order to detect an adversary trying to pass through the patrol path undetected. The challenge is to maximize the robots' chances of detecting the adversary. We model the robots and the environment, and find a family of optimal patrol algorithms for the robots. The algorithms address different adversarial models, characterized by the knowledge obtained by the adversary on the patrolling robots. Furthermore, we consider various robotic models, based on possible movement and sensing models of the robots, and different environmental models—patrolling around a perimeter (closed polygon) and fence (open polyline).

In the second part of the work, we discuss two additional canonical tasks in multi-robot systems. First, we address the problem of task reallocation in multi-robot formation, in which a subgroup of the robots should be extracted from the formation in order to perform some new task. We therefore propose

a new model of the system that is based on the interaction between the team members, which helps in finding an efficient solution to the problem. Second, we consider the problem of multi-robot coverage, in which a team of robots is required to jointly visit a target area *once*. We propose a heuristic algorithm that is shown to significantly improve the time to complete the coverage task.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The subject of multi robot systems has been thoroughly investigated over the past decade (e.g., [12, 43, 50, 61, 80, 81]). These systems have immediate applicability in a wide variety of tasks, such as military operations and space missions. Multi-robot systems are of interest for a number of reasons. For one, it may be possible to use a multiple robot system in order to accomplish tasks that a single robot cannot achieve. Other advantages of multiple robot systems have to do with the decreased cost due to the use of simpler and cheaper individual robots, and to the ability to overcome failures in the system (robustness).

Building algorithms for multi-robot systems is a complex task, for which solutions and methods are brought from many different disciplines. Probabilistic methods and control theory have been used to address sensor and action uncertainty (e.g. [34, 37]). Behavior-based and BDI methods have been used to address the high dynamic nature of the unstructured environment (e.g. [12, 83]). Ad-hoc networks and P2P methods have been used to address communication difficulties (e.g. [65, 79]). While all of these provide important infrastructures for solving problems in multi-robot systems, they usually do not formally provide proven guarantees for the performance of the proposed algorithms.

There is significant interest in multi-robot systems also in the discipline of theoretical computer science, mainly by using distributed systems analysis. The great advantage in using these methods is the ability to provide *guaranteed* characteristics in solutions to problems in the systems, for example guaranteed correctness, guaranteed performance or robustness to robot failure. An example can be found in the problem of multi-robot formation, in

which Suzuki and Yamashita [81] analyze the ability (or inability) of a team of robots to stabilize in some formation. Flocchini et al. [36] continued this line of research to examine these guarantees under different synchronization models of the robots, and Agmon and Peleg [6] analyze the problem of robot gathering under several models of robotic failures.

However, in work on multi-robot systems from a distributed algorithms perspective, researchers tend to make unrealistic assumptions about the robots' capabilities, thus their models are impractical for realistic systems. For example, robots are sometimes assumed to have unlimited visibility [6, 19], or limited radius of visibility but across 360° [11], where in practice this is usually not the case. On the other hand, robots are sometimes assumed to be extremely weak, specifically have unrealistically limited memory or computational power [78, 87].

In our work, we were motivated by methods from theoretical computer science, yet we incorporated realistic assumptions on the robots' capabilities. We chose to investigate all groups of problems using the same general method. When considering a general problem, we first extracted a basic, generic problem and modeled it in a way that can later be generalized to more sets of problems. Then, we placed theoretical foundations, for example a basic algorithm, for any possible solution based on the model. Thereafter, we used these foundations to solve more problems in the group. By applying this method we were able to guarantee system performance (e.g. probability of penetration detection in multi-robot patrol), time complexity (polynomial in multi-robot patrol and reduced time complexity, yet exponential, in task reallocation), and robustness (in multi-robot coverage).

In the first and the main part of the work we consider the problem of multi-robot patrol in adversarial environments. In this problem, a team of robots is required to continuously visit some target area in order to detect an adversary trying to pass through the patrol path undetected. We provide a model of the environment and the robots, and use this model in order to find *optimal* patrol strategies for the robots, such that their chances of detecting the adversary are maximized. In order to model the system and find the optimal solutions, we use theoretical tools such as Markov chains and dynamic programming. We further adapt the system to more realistic capabilities of the robots, specifically various sensing capabilities. In addition, we examine the impact of the adversarial knowledge on the choice of optimal patrol algorithms, and provide theoretical solutions for adversaries obtaining no knowledge and full knowledge on the patrol. In reality, since

the adversary's knowledge lies somewhere on the knowledge continuum — between full and zero knowledge — we broadly discuss this case as well , providing both theoretical and empirical evaluation of possible solutions.

In the second part of the work, we discuss two additional problems in multi-robot systems. First, we address the problem of task reallocation in multi-robot formation. In this problem, given the formation and the cost of sensing each robot by its peers, we wish to reallocate k robots for a new task while optimizing some utility function of the team. We focus on extracting the k team members while minimizing the sensorial cost of the remaining formation. We then generalize the basic problem while taking more details into consideration, for example, adding more components to the utility function. Second, we consider the problem of multi-robot coverage. In this problem, the team of robots is required to jointly visit a target *once*. In our work, we focus on creating more efficient robust paths for the robots, such that it will dramatically improve the coverage time. We provide a heuristic algorithm for constructing the paths for the robots, and via simulations we show that this heuristic significantly improves the coverage time compared to previously used algorithms.

Below we discuss the contributions of the dissertation in more detail. In Section 1.1 we describe our contributions to the problem of multi-robot patrol in adversarial environments. In Section 1.2 we introduce our work on task reallocation in multi-robot formation, and in Section 1.3 we summarize our contributions in improving efficiency in the problem of multi-robot area coverage.

1.1 Multi-Robot Patrol in Adversarial Environments

The problem of multi-robot patrol has gained interest in recent years [8, 10, 16, 30, 63], mainly due to its applicability in various security applications. In this problem, robots are required to repeatedly visit a target area, in order to monitor it. Many researchers (e.g. [16,30]) have focused on a frequency-based approach, guaranteeing that some point-visit frequency criteria are met by the patrol algorithm.

In this work, we advocate an adversarial approach in which the robots patrol in an adversarial environment, where their goal is to patrol in a way

that maximizes their chances of detecting an adversary trying to penetrate through the patrol path.

As opposed to frequency-driven approaches, in adversarial settings the frequency criteria becomes less relevant. Consider the following scenario. We are given a cyclic fence of a length of 100 meters and 4 robots must patrol around the fence while moving at a velocity of $1m/sec$. Clearly, the optimal possible *frequency* of visits at each point around the fence is 1/25, i.e., each location is visited once every 25 seconds. This is achieved if the robots move deterministically. Assume that it takes an adversary 20 seconds to penetrate the area through the fence. If the robots move in a deterministic path, then the adversary can guarantee penetration if it simply enters through a position that was currently visited by the patrolling robot. On the other hand, if the robots move non-deterministically, then the choice of penetration position becomes less trivial.

We follow our main approach of adapting theoretical computer science tools by first representing the system using a general model (in our case basically a Markovian model). We then solve the problem using tools such as a dynamic programming inspired algorithm, and finally we adapt our model and solution to various movement and sensing capabilities of the robots, while working with different environmental and adversarial models.

In this work we focus on the problem of multi-robot patrolling along a linear path - around a closed polygon (perimeter) or an open polyline (fence). We consider several possible robotic models, based on the robots' movement characteristics and on their sensing abilities. Specifically, the robots can have directionality associated with their movement (and turning around could cost the system time), or they can be omnidirectional. Their sensing abilities can be perfect or imperfect.

We then focus our research on finding optimal patrol algorithms for the robots in different adversarial models. The adversarial model is characterized by the knowledge obtained by the adversary on the patrolling robots.

We examine the case in which the adversary has full knowledge of the environment, and uses this information in order to maximize its chances of penetrating without being detected. It is therefore assumed that the adversary will penetrate through the weakest spot of the path, hence the goal of the robots is to maximize the probability of penetration detection in the weakest spot. We provide a *polynomial time* algorithm to find the *optimal* patrol for the robots in this strong adversarial environment. We show that a non-deterministic patrol algorithm is advantageous, and guarantees some

lower bound criteria on the performance of the robots, i.e., on their ability to block the adversary.

On the other hand, if the adversary has no knowledge of the patrol scheme, then the patrol scheme might be different. In this case, we assume the adversary chooses its penetration spot at random with a uniform probability. The robots, in contrast, wish to maximize the *expected* probability of penetration detection along the entire patrol path. We show the surprising result that as opposed to the sophisticated algorithms used when the adversary has full knowledge of the patrol scheme, in perimeter patrol, the optimal algorithm for the robots is the simple deterministic patrol in which the robots simply follow their patrol path without ever turning around. We show that this result does not hold for fence patrolling.

Realistically, most adversaries would have neither perfect knowledge nor zero knowledge, but *partial knowledge*. Unfortunately, optimal algorithms for either extreme case fail in partial-knowledge cases. Therefore we also discuss the case of an adversary that lies somewhere on the knowledge continuum, between full and zero knowledge, i.e., having *some knowledge*. In this case our focus is twofold. First, we provide heuristic algorithms to handle such adversaries. Second, we discuss this case theoretically, and focus on the influence of the adversary's partial knowledge on its uncertainty of its choice of action, and the impact of this uncertainty on the robots' optimal patrol algorithm.

In order to evaluate the behavior of people in this scenario with different amounts of information, we created the *Penetration Detection Game* (PenDet-Game game). In this game, simulated robots execute different patrol schemes while patrolling around a perimeter, trying to detect penetrations. The player plays the role of the adversary, hence she is required to choose a point through which, to her understanding, she will most likely penetrate without being detected by the robots. We performed two phases of the experiment. In the first phase, we focused on comparing between the performance of algorithms meant for full and zero knowledge adversaries, as well as one heuristic algorithm for adversary with some knowledge. In the second phase we focused only on adversary with some knowledge, and compared between the performance of several possible patrol algorithms meant for this case. In this work we present a detailed description of the game, its settings and the experimental results.

1.2 Task Reallocation in Multi-Robot Formation

This part of our work involves a team of N robots engaged in a cooperative task. Specifically, we consider the problem of choosing k team members in order to reassign them to a new task. We assume that all members are capable of participating in the execution of the new task, and the cost of the new task does not depend on the identity of the robots chosen to perform it. Therefore this work concentrates on the problem of choosing k robots such that the performance of the existing task, performed by the remaining group members, will be as efficient as possible.

The general problem of choosing k out of N team members in order to perform a new task such that some mutual group-objective function is optimized is an important problem. However, it was proven to be $\mathcal{NP}$-hard as a special case of the Set Partitioning Problem [41].

The measure we propose in our work, according to which the reallocation is done,is based on the interaction between team members. Our approach, which concentrates on the minimization of the cost of interaction between the entities, reduces the number of possibilities for optimal assignment, thus lessens the search domain from, theoretically, order N^k to order 2^k and makes it feasible to solve the problem both optimally and more efficiently (especially for relatively small k's).

Also in this work, we adapt our general method of first representing the system using a general model (a directed graph in this case), and then presenting a basic algorithm for solving the problem over this basic model. We also propose other adaptations of the algorithm for different scenarios, for example taking more factors into consideration in the task reallocation problem, and different domains aside from multi-robot formation.

In the model we propose, the set of team members and the interaction between them is represented by a weighted directed graph, where the vertices represent the members, the directed edges represent the interaction (interaction of a vertex a to vertex b could be different from interaction of b to a, thus the edges are directed) and edge weights represent the cost of the interaction between them. We assume the team has a leader, which can correspond to a formation leader in the example accompanying us throughout the work. In the general case, this leader is a team member that has only incoming edges and no outgoing edges, i.e., it does not depend on other team member's input

in its interaction with them.

The problem used to illustrate methods is the fundamental problem of maintaining a formation by a team of robots; this problem has received considerable attention in the literature (e.g. [12, 51]). In order to maintain the formation, a robot has to monitor one or more robots in the formation. Several common methods for choosing the identity of the robot(s) to be monitored are known [12]. We choose to focus on the method proposed in [51], in which the identity is driven by minimization of the monitoring cost of the robots. Therefore the graph is the monitoring graph which represents the sensorial capabilities of the robots, i.e., which robot can sense the other and at what cost. The problem thus involves extracting k robots in order to perform a new task (for example acquire a new target) while minimizing the monitoring cost of the remaining group.

In particular, in this work we make the following contributions.

1. We introduce a new method in which the problem of choosing k out of N team members is modeled by a graph, and the decision is taken while emphasizing the cost of interaction between the members.

2. We describe a deterministic algorithm for choosing k team members while minimizing the cost of interaction between the members of the remaining group assuming that the group has one possible leader. The algorithm is exponential in k, hence polynomial if we assume that $k = \mathcal{O}(\log N)$.

3. We later generalize the algorithm for the case in which the group has more than one member which can potentially act as the leader. These cases are significantly more complex, as they require that we check possible leader replacements; yet we show that they can still be done optimally by applying an algorithm that works in reduced time (exponential in k, hence polynomial for $k = \mathcal{O}(\log N)$). We also show that the basic algorithm can be used as a base for cases in which not all team members can be chosen for the new task. We show that in some cases the basic algorithm can still be used under these circumstances.

4. We show examples for cases in which the basic algorithm can be used with a utility function that has more than one component. In particular, we consider weighted components and prioritized components of the utility function.

5. We show that the method we use, and the basic algorithm within it, is a general method that can be used in several other domains.

We then present an empirical evaluation of the algorithm, using the Player/Stage simulated environment [44]. This implementation illustrates the use of three different task reallocation algorithms—the basic algorithm, and two algorithms that take other considerations into account (weighted and prioritized components)—in different formations.

1.3 Improving Efficiency in Multi-Robot Area Coverage

The general problem of covering an area by single or multi robot systems is a fundamental problem in robotics. This problem has applications in various domains, from humanitarian missions such as search and rescue and de-mining, to agriculture applications such as seeding or harvesting, to, recently, household cleaning. The problem has been extensively investigated in both single-robot domains (e.g. [20,48,49]) and multi-robot systems (e.g. [46,67,68,88]). The major advantages of using multi-robot systems are twofold: increasing efficiency, in our problem decreasing the coverage time, and robustness, in the sense that the system remains operative even when some robots fail. In this paper we address both advantages of the multi-robot systems in the coverage task.

This work discusses the problem of building *efficient* coverage paths for a team of robots. In this problem, a team of robots, each equipped with some tool, for example a sensor, are required to jointly sweep the entire given terrain while minimizing the total coverage time. In our work, we assume that the area is divided into cells, and the robots travel through all cells of the terrain. Following the taxonomy presented in [18], the division of the area is an approximate cellular decomposition, and we handle both online and offline coverage. In offline coverage, upon which we focus, the map of the area is given in advance, therefore the coverage paths of the robots can be determined prior to execution of the coverage algorithm.

In this work we follow our general theoretical computer science inspired method (described in the general section of the Introduction), however we base our work upon a common model of representation of the system (using graphs drawn over the approximate cell decomposition of the area), in

contrast to creating a new model as presented in the previous sections. We propose a basic algorithm for improving efficiency of the coverage task in this model, and examine several different measures using the basic algorithm.

Often, previous work has pointed out that one advantage of using multiple robots for coverage is the potential for more efficient coverage [18]. Another potential advantage of using multiple robots is that they may offer greater *robustness*: Even if one robot fails catastrophically, others may take over its coverage subtask. In other words, as long as there one non-faulty robot exists, the coverage mission will be completed successfully. Unfortunately, this important capability has been neglected in most existing work on on-line algorithms.

Several methods can be found in the literature on coverage by single and multi-robot systems. One basic method, which received considerable attention, is the technique presented by Gabriely and Rimon [39], where the authors describe a polynomial time *Spanning Tree Coverage* algorithm, better known as the STC algorithm. In their work, Gabriely and Rimon offer a method for finding a Hamiltonian cycle covering a terrain that satisfies some assumptions. In particular, it is assumed that the robot is equipped with a square shaped tool of size D, hence the area is divided into N cells of size D placed on a grid. The grid is then made coarse such that each new cell is of size $2D\ X\ 2D$, and a spanning tree is built according to this new grid. After such a tree is built, the robot follows the tree around, creating a Hamiltonian cycle and visiting all cells of the original grid. The idea was first generalized for a multi-robot system in [46], with the family of MSTC algorithms. A different variation of this idea was introduced in [88].

When building the tree in a single robot system, the influence of the structure of the tree is theoretically irrelevant to the coverage time. Clearly, one might want to construct spanning trees with special characterizations, for example minimizing the number of turns of choosing some preferred directionality. Nonetheless, the coverage time guaranteed by the STC algorithms is linear in the size of the grid, since each cell, except for the boundary cells, is covered once, hence the total coverage time is N.

On the other hand, in multi-robot systems, the structure of the tree can have a crucial impact on the coverage time of the terrain. The choice of the spanning tree can change the robots' initial positions with respect to each other, from being concentrated, i.e., placed as a bundle, to being scattered along the spanning tree path — all without actually changing the physical initial position of the robots. The structure of the tree itself can therefore

substantially reduce the coverage time required by algorithms based upon it. Hence we concentrate on building appropriate coverage spanning trees. The general method we follow when building such trees, is to gradually grow subtrees from the initial position of the robots.

Hence, we deal with constructing spanning trees for offline coverage that reduces the total coverage time of algorithms using them as a base for coverage. The coverage time of a terrain is determined by the robot traveling through the terrain longest period of time. In a system with homogenous robots, this time corresponds to the longest distance traversed by a single robot. We try to minimize this distance by creating trees where the robots are placed as uniformly as possible around it. Therefore, when constructing the trees we try to minimize the maximal distance between every two consecutive robots along the spanning tree path. If such a tree is obtained, we show that all versions of the MSTC algorithm that ran on these trees achieve substantially better coverage time compared to their coverage time on other randomly generated trees. Note that these trees, along with decreasing the coverage time of the algorithms which use them as a base for coverage, also enjoy the benefits of the algorithms themselves. Specifically, if used as a base for the family of MSTC algorithms, it guarantees robustness. The tree construction algorithm we propose herein was tested both in a "clean" terrain, i.e., without obstacles, and also terrains with obstacles, where the obstacles are assumed to occupy a full cell (or cells) of the coarse grid. The algorithm we propose has a polynomial time complexity in the number of cells to be covered. This results in the surprising conclusion that as we add obstacles to the terrain, the complexity of the tree construction algorithm lessens, since the number of covered cells diminishes.

1.4 Thesis Overview

This dissertation comprises 9 chapters, organized into two main parts (see Figure 1.1). This chapter constitutes the introduction to this thesis. The next chapter surveys the related work. Chapters $3--6$ constitute Part 1 of the dissertation, which deals with multi-robot patrol in adversarial environments. Part 2 of the dissertation includes other problems in the multi-robot field, where Chapter 7 addresses the problem of team member reallocation in multi-robot formation and Chapter 8 deals with the problem of multi-robot coverage. In Chapter 10 we provide our conclusions and discuss future work.

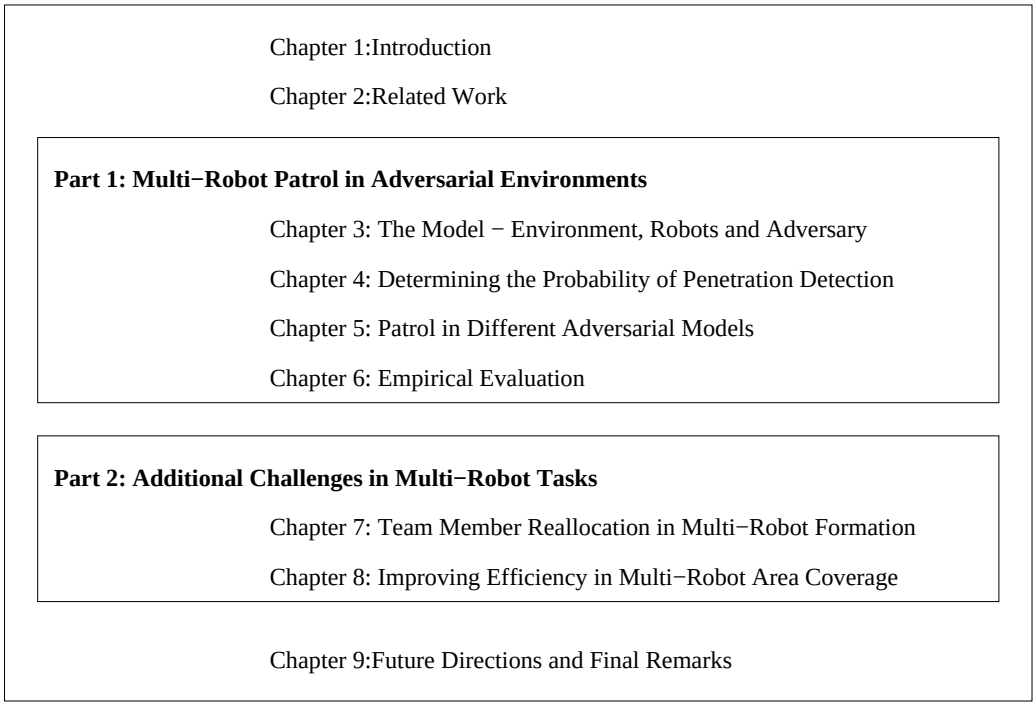


Figure 1.1: Thesis Structure.

1.5 Publications

Results that appear in this dissertation have been published in the proceedings of the following refereed journals, conferences, books and workshops:

- Noa Agmon, Noam Hazon and Gal A. Kaminka. The Giving Tree: Constructing Trees for Efficient Offline and Online Multi-Robot Coverage. In a special issue of the Annals of Math and Artificial Intelligence (AMAI) Journal on Multi-Robot Coverage, Search, and Exploration, 2009, in press. [58]

- Noa Agmon, Sarit Kraus and Gal A. Kaminka. Multi-Robot Fence Patrol in Adversarial Domains. In Proc. of the 10th Conference on Intelligent Autonomous Systems (IAS-10). IOS Press, July 2008. [4]

- Noa Agmon, Sarit Kraus and Gal A. Kaminka. Multi-Robot Perimeter Patrol in Adversarial Settings. In Proc. of IEEE International Conference on Robotics and Automation (ICRA 08). May 2008. [5]

- Noa Agmon, Vladimir Sadov, Sarit Kraus and Gal A. Kaminka. The Impact of Adversarial Knowledge on Adversarial Planning in Perimeter Patrol. In Proc. of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2008). May 2008. [7]

- Noa Agmon, Noam Hazon and Gal A Kaminka . Constructing Spanning Trees for Efficient Multi-Robot Coverage. In Proc. of IEEE International Conference on Robotics and Automation (ICRA 06). May 2006. [2]

- Noa Agmon, Gal A Kaminka and Sarit Kraus. Team Member-Reallocation via Tree Pruning. In Proc. of the National Conference on Artificial Intelligence (AAAI), pages 35-40, July 2005. [3]

In addition, the following papers were both influenced by and reflected in this work. are discussed in the related work section, though their results *are not* part of this thesis.

- Yehuda Elmaliach, Noa Agmon and Gal A. Kaminka, Multi-Robot Area Patrol under Frequency Constraints, Annals of Math and Artificial Intelligence, 2009, in press. [32]

- Yehuda Elmaliach, Noa Agmon, and Gal A. Kaminka. Multi-Robot Area Patrol under Frequency Constraints. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA-07), 2007 [30].

Chapter 2

Related Work

2.1 Multi-Robot Patrol

Systems comprising multiple robots or agents cooperating in order to patrol in some designated area have been studied based on various approaches and in different contexts. Theoretical and empirical solutions have been proposed in order to assure quality patrol. The definition of quality depends on the context. Most studies concentrate on the frequency of visits throughout the designated area. Efficient patrol, in this case, is a patrol guaranteeing a high frequency of visits in each part of the area. If the robots work in an adversarial environment, then efficient patrol is one that deals efficiently with intruders.

Note that many of these related papers were published *after* our initial work, which was presented in [5, 7].

The first theoretical analysis of the multi-robot patrol problem was presented by Chevaleyre [16]. He introduces the notion of *idleness*, which is the duration each point in the patrolled area is not visited. In his work, he analyzes two types of multi-robot patrol schemes with respect to the idleness criteria: partitioning the area into subsections, where each section is visited continuously by one robot; and the cyclic scheme in which a patrol path is provided along the entire area and all robots visit all parts of the area, consecutively. He proves that in the latter approach, the frequency of visiting points in the area is considerably higher. Another survey by Almeida et al. [9] offers an empirical comparison between different approaches towards patrolling with regards to the idleness criteria, and shows great advantage of

the cycle based approach.

Elmaliach et al. [30,31] offer new frequency optimization criteria for evaluating patrol algorithms. They provide an algorithm for multi-robot patrol that is proven to have optimal frequency as well as uniform frequency, i.e., each point in the area is visited with the same highest-possible frequency. Their work is based on creating one patrol cycle that visits all points in the area in minimal time, and the robots simply travel equidistantly along this patrol path.

Sak et al. [69] considered the case of multi-agent patrol in adversarial environments in general graphs (rather than perimeters, as in our work). They concentrated on an empirical evaluation (using a simulation, with no human subjects involved) of several non-deterministic patrol algorithms that can be roughly divided into two: Those that divide the graph between the patrolling agents, and those that allow all agents to visit all parts of the graph. They considered three types of adversaries: random adversary, an adversary that always chooses to penetrate through a recently-visited node and an adversary that uses statistical methods to predict the chances that a node will be visited soon. They concluded that there is no patrol metric that outperformed the others in all the domains they have checked, but the optimality depends on the environment. In contrast to this investigation, we provide empirical results from tests with human subjects, and theoretic proofs of optimality for different settings.

Other closely related work is the work by Paruchuri et al. [63], which considered the problem of placing security checkpoints in adversarial environments. Similar to our assumptions, they assume that their agents work in an adversarial environment in which the adversary can exploit any predictable behavior of the agents. They use policy randomization in the agents' behavior in order to maximize their rewards. They assume the adversary has *full knowledge* of the patrolling agents. Paruchuri et al. further study ([62]) this problem in cases where the adversarial model is unknown to the agents, although the adversary still has full knowledge of the patrol scheme. They again provide heuristic algorithms for optimal strategy selection by the agents. In our work, we discuss different adversarial models determined by the extent of information revealed to the adversary. Also, we assume the robots have partial information about the adversary (specifically, they know the time it takes the adversary to penetrate).

Pita et al. [64] continued this research to consider the case in which the adversaries make their choices based on their bounded rationality or uncer-

tainty, rather than make the optimal game-theoretic choice. They considered three different types of uncertainty over the adversary's choices, and provide new algorithms that deal with these types of uncertainties. In our work we discuss other aspects of uncertainty in an adversary's choice, and provide *optimal polynomial-time* solutions.

Amigoni et al. [10] also used a game-theoretic approach for determining the optimal strategy for patrolling agents, using leader-follower games. They consider an environment in which a robot can move between any two nodes in a graph, as opposed to the perimeter model we use. Their solution is suitable for one robot, and since the computation of the optimal strategy is exponential, they described a heuristic approach for finding a solution.

Theoretical work based on stochastic processes that is related to our work is the *cat and mouse* problem [21], also known as the *predator-prey* [45] or *pursuit evasion* problem [85]. In this problem, a cat attempts to catch a mouse in a graph where both are mobile. The cat has no knowledge about the mouse's movement, therefore as far as the cat is concerned, the mouse travels similarly to a simple random walk on the graph. We, on the other hand, have worst case assumptions about the adversary. We consider a *robotic* model, in which the movement is correlated to the movement of a robot, with possible directionality of movement and possible cost of changing directions. Moreover, in our model the robots travel around a perimeter, rather than in a graph or an area.

Other theoretical work by Shieh and Calvert [77], based on computational geometry solutions, attempts to find optimal viewpoints for patrolling robots. They try to maximize the view of the robots in the area, show that the problem is $\mathcal{NP}$-Hard, and find approximation algorithms for the problem.

2.2 Task Reallocation and Multi-Robot Formations

The class of pattern-formation problems of multi robot systems is discussed in [12,38,59,81]. Balch and Arkin [12] describe three possible basic techniques a robot can use in order to maintain its position in a formation: leader referenced, neighbor referenced and unit-center referenced. Further studies by Desai [25], Lemay et al. [56] and Kaminka et al. [51] examine different aspects of the neighbor referenced technique. In our work, we consider the

situation in which the formation, built according to the method proposed by Kaminka et al. [51], already exists, and we need to extract k team members from the formation. Our problem does not resemble the problem where agents fail to operate [55], since we are able to choose the k members, which results in maximization of the actions of the remaining team members.

Our problem belongs to the ***multi robot task allocation*** (MRTA) domain. Following the taxonomy for MRTA systems given by [43], our work deals with instantaneous assignments of single-task robots performing multi-robot tasks.

Studies that discuss the problem of choosing k out of N agents in order to perform a new task mostly concentrate on maximizing the profit gained by the optimal performance of the new task. For example in [42] Gerkey and Matarić used the publish/subscribe messaging paradigm to assign robots to a given task. We, on the other hand, concentrate on maximizing the benefit (minimizing the cost) gained by the optimal execution of the original task. The problem can be considered equivalent, if it is perceived as choosing $N - k$ team members to perform the original task. Nevertheless, we aim to find the exact optimal solution where in other studies the problem is handled heuristically.

Other studies discuss allocation of agents to several given tasks. These studies mostly provide heuristic algorithms for efficient allocation of agents to tasks. In [70], Sander et al. describe task allocation heuristic algorithms for settings in which the agents and tasks are geographically dispersed in the plane. Work presented in [27, 28] discusses task allocations between robots using auctions. Finding the optimal assignment using combinatorial auctions, where bidders can bid on combinations of items, has been proven to be $\mathcal{NP}$-hard [71].

The problem of designing and forming groups of agents while maximizing some mutual objective is usually referred to as coalition formation. The work of Shehory and Kraus [76] is close to the scenario discussed in this paper. They suggest algorithms for coalition formation among agents, where an agent can be either a member of only one coalition (similar to our case), or coalitions may overlap. They provide heuristic algorithms, rather than an exact optimal solution. In [73], Sandholm and Lesser also dealt with coalition formation, but with self interested agents. Our problem can be perceived as a private case of the general coalition formation studies, for example two tasks - new and old or two coalitions. In [84], Tosic and Agha describe a distributed algorithm for generating coalitions based on the current physical

configuration of the agents, using maximal cliques. They show that the agents convert to the same coalitions, but their work does not refer to any kind of group utility, as opposed to our work, which maximizes the joint utility of the agents performing the original task.

Another subject in coalition formation which is closely related to the problem discussed here is the problem of coalition structure generation [24, 72, 74]. In this problem, which has been shown to be $\mathcal{NP}$-hard, a division of the agents into coalitions is searched such that the group utility (payoff) is maximized. In [24, 72], the authors provide algorithms with a guaranteed lower bound from the optimal. In [74], Sen and Dutta describe a stochastic search approach for searching for optimal coalitions. As opposed to our scenario, whereby an optimal solution is guaranteed, in these studies a sub optimal solution is given.

2.3 Multi-Robot Coverage

The challenge of covering a terrain by a team of mobile robots has received considerable attention in the literature. The growing interest in this area is first and foremost due to the fact that the coverage task is implementable in various domains. Moreover, the concentration in multi-robot systems comes from two key features made possible by using multiple robots: (i) robustness in the face of single-robot catastrophic failures, and (ii) enhanced productivity, thanks to the parallelization of sub-tasks. Many approaches can be found in the literature for multi-robot coverage.

Choset [18] provides a survey of coverage algorithms, which distinguishes between *offline* algorithms, in which a map of the work-area is given to the robots in advance, and *online* algorithms, in which no map is given. The survey further distinguishes between *approximate cellular decomposition*, where the free space is approximately covered by a grid of equally-shaped cells, and *exact decomposition*, where the free space is decomposed to a set of regions, whose union fills the entire area exactly. Following Choset's terminology, in this paper we focus on both online and offline coverage, based on approximate cell decomposition of the area.

We focus on spanning tree based coverage, first proposed by Gabriely and Rimon in [39]. They proposed the basic method of dividing the terrain into $2D\ X\ 2D$ cells, and described the polynomial time spanning tree coverage algorithm (STC) for complete offline and online coverage of the terrain by a

single robot. In [40], they suggest two different algorithms for building an on-line tree, but the motivation comes from the desire to create a spanning tree with a specific scanning direction.

The generalization of the single-robot STC algorithm to offline multi-robot systems was first introduced by Hazon and Kaminka in [46]. They presented several offline algorithms for multi-robot coverage of a terrain by the MSTC algorithm, which guarantees robust, time-efficient and complete coverage. They describe two versions of the MSTC algorithm: non-backtracking MSTC, and backtracking MSTC, herein referred to as NB_MSTC and B_MSTC, respectively. In the NB_MSTC algorithm the robots simply move in a counterclockwise direction along the spanning tree path until they reach the initial position of the following robot if no faults occur, or otherwise take over the coverage path of the consecutive robot. In the B_MSTC the robots can backtrack over parts of their coverage path, i.e., they can go both clockwise and counterclockwise. They have shown that if the robots backtrack, the worst case performs up to twice as fast as the non-backtracking case, despite the redundancy. Other results by Hazon and Kaminka, described in [47], provide an optimal polynomial time coverage algorithm, herein referred to as Opt_MSTC. Opt_MSTC is similar to the B_MSTC algorithm with modifications that assure optimal coverage time given the initial locations of the robots and an initial spanning tree. The optimality is guaranteed only for the backtracking method, i.e., if the robots go back and forth *along the given spanning tree*. Hence they promise to make the most (optimal) out of the given tree and initial locations of the robots if the robots do not deviate from the path dictated by the structure of the tree. In our work we focus on generating good trees for such algorithms.

Work by Zheng et al. [88] proposed an additional offline multi-robot coverage algorithm. Their solution is based on dividing the *given* spanning tree into k subtrees, where a path overlapping between robots might exist. Their algorithm performs better compared to both NB_MSTC and B_MSTC algorithms, however their solution is *not* robust. In addition, they note that different choices of trees may result in different coverage time, but they did not expand on this issue.

There have been additional investigations of online multi-robot coverage, for example in the world of ant robotics. Wagner at. al. [86] propose a series of theoretical multi-robot ant-based algorithms which use approximate cellular decomposition. The algorithms involve little or no direct communications. Instead they use simulated pheromones for communication or traces

of robots. Some of these algorithms solve only the discrete coverage problem and some offer complete robust coverage, but not necessarily efficient. Recent work by Osherovich et al. [60] offers a robust coverage algorithm for ants in continuous domains. Svennebring and Koenig [82] offer a feasibility study for ant coverage. They perform experiments with real ant-robots in large-scale simulations. They show robustness, but provide no analytic guarantees for completeness or efficiency.

Acar and Choset [1] present a robust on-line *single* robot coverage algorithm while their robustness quality is the ability to filter bad sensor readings.

Rekleitis et al. [66] use two robots in online settings, with a visibility graph-like decomposition (more or less an exact cellular decomposition). The algorithm uses the robots as beacons to eliminate odometry errors, but does not address catastrophic failures (i.e., when a robot dies). In a more recent article, Rekleitis et al. [68] extend the Boustrophedon approach [18] to a multi-robot version. Their algorithm also operates under the restriction that communication between two robots is available only when they are within line of sight of each other. Their solution, though, is not robust to failures, i.e., it could stop functioning if one of the key robots fails. In [53], Kong et al. provide an improved algorithm for multi-robot coverage with unbounded communication, where the algorithm is demonstrated to be robust to failures (yet this property is not theoretically proven to be complete).

Butler et al. [15] propose a sensor-based multi-robot coverage, in a rectilinear environment, based on the exact cellular decomposition. They do not prove their robustness, and the robots could cover the same area many times.

The recent Brick & Mortar algorithm suggested by Ferranti et al. [35] is an online coverage algorithm that assumes the robots communicate using miniature storage devices that are placed along the entire area, such that one device is placed in each cell. The use of these devices is their solution to the extensive communication assumption, made by other online coverage algorithm, which is partially made in our work as well. Their work does not refer to the robustness of the coverage. In addition, their solution might result in redundancy of coverage.

Other approaches, other than the ones based on cellular decomposition of the terrain, can be found in literature on multi-robot coverage. For example, in [13], Batalin and Sukhatme offer two coverage algorithms for a multi-robot system in which the robots spread out in the terrain, and move away from each other while covering the area and minimizing their interaction. In

their work, they aim to achieve optimal coverage *area*, and do not prove any formal statement regarding the optimality of coverage time. Yet, similar to their work, our tree construction algorithm uses the "spreading out" principle in order to build the coverage tree.

Part I

Multi-Robot Patrol in Adversarial Environments

In this part we consider the problem of multi-robot patrol with the existence of an adversary attempting to penetrate through the patrol path without being detected. We develop a *non-deterministic* patrol framework for the robots, such that their movement is characterized by a probability p. We model different environments—patrol around a closed polygon, perimeter, and patrol along an open polyline, fence. We use this framework to also handle different robotic models, specifically robots with different movement characteristics (directed or undirected movement) and different sensing capabilities (perfect or imperfect, local or extended).

We then examine the impact of the knowledge obtained by the adversary on the choice of patrol algorithm. This work explores this question in depth and provides theoretical results, supported by extensive experiments concerning the compatibility of algorithms to the extent of information possessed by the subjects. First, we analytically examine the case of a full-knowledge adversary, and offer an optimal polynomial-time algorithm in order to find the optimal patrol algorithm such that the minimal probability of penetration detection throughout the perimeter is maximized. We then address a zero-knowledge opponent—a different extreme—and show that surprisingly, this seemingly best-case scenario (from the point of view of defending robots) is optimally addressed by a deterministic, non-randomizing patrol for perimeter patrol. Nonetheless, this result does not hold for fence patrol. We provide a discussion of the case in which the adversary has *some* knowledge (between full and zero knowledge), and provide both heuristic algorithms and optimal algorithms that depend on the level of uncertainty in the adversary's choice. Finally, we present results from extensive experiments using human subjects playing the role of the adversary that work opposite simulated robots, and thoroughly discuss the experimental results.

This part is organized as follows. In Chapter 3 we describe the general model - environment, robot and adversarial model, and the patrol algorithm framework. In Chapter 4 we provide algorithms for determining the probability of penetration detection in both perimeter and fence patrol, and also describe the results concerning various sensing abilities of the robots. In Chapter 5 we discuss and present optimal patrol algorithms concerning the influence of knowledge obtained by the adversary on the choice of patrol algorithm, focusing on full knowledge, zero knowledge and patrial knowledge. In Chapter 6 we comprehensively describe the experiment we conducted and analyze the results we obtained.

Chapter 3

The Model - Environment, Robots and Adversary

In this chapter we provide basic definitions concerning the assumptions on the robots' behavior and coordination and the influence of these attributes on the patrol mission.

3.1 Robot and environment model

3.1.1 Robotic computational model

We consider a system consisting of k homogenous mobile robots, that are required to patrol around a closed polygon or an open polyline P. The robots operate in cycles, where each cycle consists of two stages.

1. **Compute**: Execute the given algorithm, resulting in goal point p_G.
2. **Move**: Move towards point p_G.

This model is synchronous, i.e. all robots execute each cycle simultaneously. We consider a patrol in a circular or open path, which is similar to a one dimensional (linear) graph.

The description herein refers to patrolling around a closed polygon P. A detailed description of patrolling along an *open polyline* is given in Section 4.3.

The path around P is divided into segments of a length of l, where l corresponds to the distance one robot travels and monitors the area in one

cycle. Hence each robot travels through one segment per cycle while covering it (its velocity is 1 segment per one time cycle). This division into segments makes it possible to consider patrols in heterogeneous terrains. In such areas, the difficulty of passing through terrains varies from one terrain to another, for example driving in muddy tracks vs. driving on a road. In addition, riding around corners requires a vehicle to slow down. Figure 3.1 demonstrates a transition from a given area to a discrete cycle.

Denote the total number of segments around the perimeter by N. Note that the distance between the robots is calculated with respect to the number of segments between them, i.e., the distance is in travel time. For example, if we say that the distance between R_1 and R_2 is 7, then there are 7 segments between them, and if R_1 had remained still, then it would have taken R_2 7 time cycles to reach R_1.

Each cycle a robot that resides in segment s_i can have three options of where to go - segment s_{i-1}, segment s_{i+1} or remain in segment s_i. We assume the robots are coordinated, i.e., all robots decide simultaneously to move in the same direction. We also require that the robots are initially placed uniformly around P with a distance of $d = N/k$ between every two consecutive robots. The motivation for these assumptions is shown in Section 3.3.

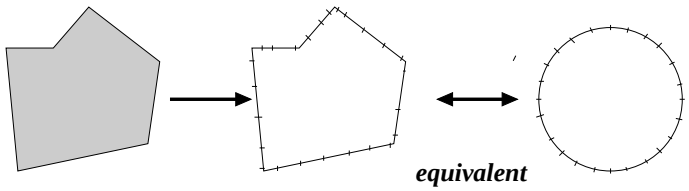


Figure 3.1: An example for creating discrete segments from a circular path with the property that the robots travel through one segment per cycle.

Definition: Let s_i be a discrete segment of a perimeter P which is patrolled by one robot or more. Then the *Probability of Penetration Detection* in s_i, ppd_i, is the probability that a penetrator going through s_i during t time units will be detected by some robot going through that segment during that period of time. In other words, ppd is the probability that a patrol path of some robot will pass through segment s_i during the time that a penetrator will go through that segment. We use the general acronym ppd when referring to the general term of probability of penetration detection (without reference

to a certain segment).

3.1.2 Robotic movement model

The execution of the patrol differs from one model to the other in the Compute step. As mentioned previously, we consider three different patrol models, based on the robots' movement abilities.

1. Bidirectional Movement Patrol ($\mathcal{BMP}$)
2. Directional Zero-Cost Patrol ($\mathcal{DNCP}$)
3. Directional Costly-Turn Patrol ($\mathcal{DCP}$)

The $\mathcal{BMP}$ patrol is intended for robots with a movement pattern similar to movement on train tracks or a camera going back and forth along a fixed course (omnidirectional robots). In this model, the robots have no movement directionality in the sense that switching directions—right to left and vice versa—does not require physically changing the direction of the robot (turning around).

In the other two models the robots' movement is directed, and turning around is a special operation that might have an attached cost in time. The $\mathcal{DNCP}$ patrol is used for robots which have directionality of movement, but turning around does not consume extra time. The $\mathcal{DCP}$ patrol model is a more general and a more realistic version of the $\mathcal{DNCP}$ model, where if the robot turns around, it remains in its current position for $\tau > 1$ time units, i.e., switching direction costs the system extra time. An example of this kind of robot is the differential drive robot commonly used in research labs.

3.1.3 Adversarial model

Our basic assumption is that the system works with the existence of an adversary that tries to penetrate through the patrolling robots without being detected. The adversary decides, in time 0, which segment to penetrate. We assume the adversary tries to penetrate *once* some segment and the robots try to detect it. Its penetration time is *not instantaneous*, and lasts t time units.

Recall that the time distance between every two consecutive robots around the perimeter is $d = N/k$. Therefore we consider t values between the

boundaries $\lfloor \frac{d}{2} \rfloor + \tau \leq t < d$. The reason for this is that if $t < \lfloor \frac{d}{2} \rfloor + \tau$, then there is at least one segment s_i with $\mathsf{ppd}_i = 0$, therefore a strong adversary will *always* manage to successfully penetrate regardless of the actions taken by the patrolling robots. On the other hand, if $t \geq d$ then *all* segments s_i can have $\mathsf{ppd}_i = 1$ simply by using a deterministic algorithm.

We consider several adversarial models. These models vary in the amount of information the adversary gained on the patrolling.

The weakest adversarial model is the model in which the adversary has *zero knowledge* on the patrol algorithm. The only knowledge it has is the current location of the robots. We assume that in this case the adversary chooses its penetration spot at random from all currently unoccupied segments, with uniform distribution

The other adversarial model is at the other extreme end of the knowledge scale, in which we assume the adversary has *full knowledge* of the patrolling robots. We define the *patrol scheme* of the robots as the

1. Number of robots, the distance between them and their current position.

2. The movement model of the robots and any characterization of their movement.

Therefore the full knowledge adversary knows the patrol scheme, and will take advantage of this knowledge in order to choose its penetration spot as the *weakest spot* of the patrol, i.e., the segment with minimal ppd. The adversary can learn the patrol scheme by observing the behavior of the robots for a sufficient amount of time. Note that in security applications, such strong adversaries exist. In other applications, the adversary models the behavior of the system in the "worst case scenario" from the patrolling robots point of view.

We also consider the case in which the adversary's knowledge lies somewhere on the knowledge continuum, between full and zero knowledge (see a full discussion in Section 5.3.

Note that we assume that the adversary will try to penetrate *once* through some segment. Also, the robots are responsible only for *detecting* penetrations and *not* handling the penetration (which requires task-allocation methods). Therefore the case in which the adversary issues multiple penetrations is similar to handling a single penetration, as the robots detect, report and continue to monitor the rest of the path, according to their algorithm.

3.2 Problem definition

The general definition of the problem is as follows.

Penetration detection (*PD*) problem: Given a circular fence (perimeter) of a length of l divided into N segments, k robots uniformly distributed around this perimeter with a distance of $d = N/k$ (in time) between every two consecutive robots, and the movement model of the robots, assume that it takes t time units for the adversary to penetrate, and the adversarial model is known. Find the optimal patrol algorithm for the robots such that it will maximize their probability of detecting the adversary.

The optimal patrol algorithm depends on the adversarial model. For example, when facing a full knowledge adversary that knows the weakest spot of the algorithm, the optimal patrol algorithm is the one that strengthens the weakest spot as much as possible, i.e., maximizes the segment with minimal ppd. On the other hand, if the adversary has zero knowledge, then the optimal algorithm is the one that maximizes the expected ppd throughout the perimeter.

3.3 Algorithm framework

Assume a robot is currently located in segment s_i. In the $\mathcal{BMP}$ model, it moves one step to the right (segment $i+1$) with a probability of p and one step to the left (segment $i-1$) with a probability of $q = 1-p$. This model is similar to a random walk. See Figure 3.2a for an illustration. In both the $\mathcal{DNCP}$ and $\mathcal{DCP}$ models, we assume directionality of movement, hence the robot continues its movement in its current direction with a probability of p, and turns around (rewinds) with a probability of $q = 1-p$. Therefore in the $\mathcal{DNCP}$ model, if the robot is facing segment $i+1$, then with a probability of p it will go straight to it and with a probability of $1-p$ it will turn around and reach cell $i-1$. Similarly, if it is facing segment $i-1$, then with a probability of p it will reach $i-1$ and with a probability of $1-p$ it will reach segment $i+1$. The $\mathcal{DCP}$ model is similar, though if the robot turns around it will remain in segment i. See Figures 3.2b. and 3.2c. for illustrations of the $\mathcal{DNCP}$ and $\mathcal{DCP}$ models, respectively.

Since the different patrol algorithms we consider vary in the probability p of the next step, we assert that the probability p *characterizes the patrol algorithm.*

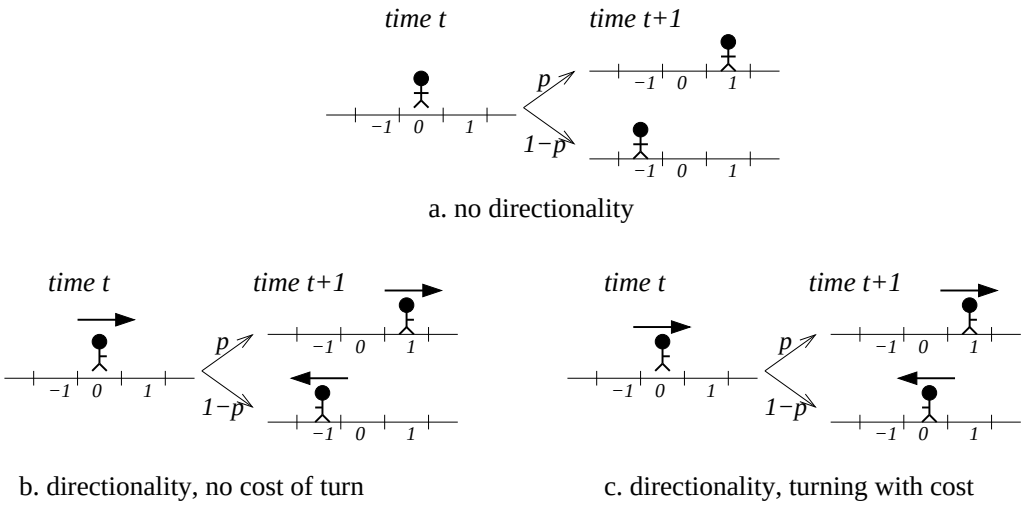


Figure 3.2: Illustration of p's characterization of the three models of movement.

Note that the probability of penetration detection in each segment s_i, $1 \leq i \leq d$, is determined by probability p characterizing the patrol algorithm. By the definition of ppd_i, we need to find the probability that s_i will be visited during t time units. Assuming perfect detection capabilities of the robots, ppd_i is determined only by the *first* visit to s_i, since once the intruder is detected the detection mission is successful. Therefore ppd_i is actually the probability that a segment will be visited *at least once* during t time units. Denote the probability of detecting a penetrator by robot R_a in segment s_j after t time units by ppd_j^a. Note that ppd_i is the sum of probabilities that $R_1, \ldots, R_k$ will visit that segment during this time unit, i.e., $\mathsf{ppd}_i = \sum_{j=1}^{k} \mathsf{ppd}_i^j$. Also, the value of ppd is calculated *regardless of the actions of the adversary* (algorithms for calculating ppd_i are described in Chapter 4).

As stated previously, we assume the robots are uniformly distributed along the perimeter with a distance of $d = N/k$ between every two consecutive robots, and that they are coordinated in the sense that if they are supposed to turn around, they do so simultaneously. In the following lemmas we prove optimality of these assumptions in both full knowledge and zero knowledge adversarial models.

Lemma 1. *For a given p, the function $\mathsf{ppd}_i^a : \mathbb{N} \Rightarrow [0,1]$ for constant t and R_a residing in segment s_0 is a monotonic decreasing function, i.e., as the*

distance between a robot and a segment increases, the probability of reaching it during t time units decreases.

Proof. We need to show that for all i, $i \in \mathbb{N}$, $\mathsf{ppd}_i^a \geq \mathsf{ppd}_{i+1}^a$. The movement of the robots is coherent, i.e., in order to move from segment i to segment $i+2$, it has to move through segment $i+1$. Therefore the probability of arriving at segment i given that it has arrived at segment $i+1$ is 1, i.e., $\mathsf{ppd}_{i|i+1}^a = 1$. By the conditional probability law, if $\mathsf{ppd}_{i+1}^a > 0$ then

$$\mathsf{ppd}_{i|i+1}^a = \frac{\mathsf{ppd}_{i+1\cap i}^a \cdot \mathsf{ppd}_i^a}{\mathsf{ppd}_{i+1}^a} = 1$$

$$\Rightarrow \mathsf{ppd}_{i+1}^a = \mathsf{ppd}_{i+1\cap i}^a \cdot \mathsf{ppd}_i^a \leq \mathsf{ppd}_i^a$$

If $\mathsf{ppd}_{i+1}^a = 0$, then since ppd's value can not be lower than 0, then inevitably $\mathsf{ppd}_i^a \geq \mathsf{ppd}_{i+1}^a$. □

Lemma 2. *As the distance between two consecutive robots along a cyclic patrol path is smaller, the* ppd *in each segment is higher and vice versa.*

Proof. Consider a sequence S_1 of segments $s_1, \ldots, s_w$ between two adjacent robots, R_a and R_b, where s_1 is adjacent to the current location of R_a and s_w is adjacent to the current location of R_b. Let S_2 be a similar sequence, but with $w-1$ segments, i.e., the distance between R_a and R_b decreases by one segment. Assume that other robots are at a distance greater than or equal to $w-1$ from R_a and R_b, and that $w-1 < t$. Since a robot may influence the ppd in segments that are up to a distance t from it (as it has a probability of 0 of arriving at any segment at a greater distance within t time units), the probability of penetration detection, ppd, in these sequences is influenced only by the possible visits of R_a and R_b.

Denote the probability of penetration detection in segment $s_i \in S_j$ by $\mathsf{ppd}_i(j)$, $1 \leq i \leq w$, $j \in \{1,2\}$, and the probability that the penetrator will be detected by robot R_l by $\mathsf{ppd}_i^l(j)$. Therefore, for any segment $s_i \in S_j$, $\mathsf{ppd}_i(j) = \mathsf{ppd}_i^a(j) + \mathsf{ppd}_i^b(j)$. Note that either $\mathsf{ppd}_i^a(j)$, $\mathsf{ppd}_i^b(j)$ or both can be equal to 0. We need to show that $\mathsf{ppd}_i(2) \geq \mathsf{ppd}_i(1)$, for all $1 \leq i \leq w$, and for at least one segment s_m, $\mathsf{ppd}_m(2) > \mathsf{ppd}_m(1)$.

First, for s_w, $\mathsf{ppd}_w(2) = 1$ as R_j is presently in segment s_w of S_2. Therefore $\mathsf{ppd}_w(2) = 1 \geq \mathsf{ppd}_w(1)$.

For every other segment s_i, $\mathsf{ppd}_i^a(j)$ remains the same (there is no change in its relative location), hence we need to examine the change in $\mathsf{ppd}_i^b(j)$.

From Lemma 1 we know that $\mathsf{ppd}_i^b(j)$ is a monotonic decreasing function. Therefore for each i, $\mathsf{ppd}_i^b(2) \geq \mathsf{ppd}_i^b(1)$. We need to show that for at least one segment $\mathsf{ppd}_i^b(2) > \mathsf{ppd}_i^b(1)$. A robot may influence the ppd on both of his sides - segments located to the left and to the right of its current position. Denote the number of influenced segments to its right by s (s may be equal to 0). If $s > 0$, then $\mathsf{ppd}_{w-s+1}^b(2) > \mathsf{ppd}_{w-s}^b(1)$. In other words, R_b has a probability of 0 of reaching the segment with a distance of $s+1$ from it in S_1, but in S_2 it is s segments away from it, therefore R_b has a probability greater than 0 to reach it. If $s = 0$, then $\mathsf{ppd}_w^($2) = 1 > \mathsf{ppd}_w(1)$, as R_b lies exactly in segment s_w of S_2, and $\mathsf{ppd}_k^b(1) = 0$. □

Theorem 3. *A team of k mobile robots engaged in a patrol mission maximizes minimal* ppd *if the following conditions are satisfied.* **a.** *The time distance between every two consecutive robots is equal* **b.** *The robots are coordinated in the sense that they constantly move together in the same direction.*

Note that condition **b** means that all robots move together in the same direction, i.e., if they change direction, then all k robots change their direction simultaneously.

Proof. Following Lemma 2, it is sufficient to show that the combination of conditions **a** and **b** yield the minimal distance between two consecutive robots along the cyclic path. Since we have N segments and k robots, there are $\binom{N}{k}$ possibilities of initial placement of robots along the cycle (robots are homogenous, so this is regardless of their order). If the robots are placed uniformly along the cycle, then the time distance between each pair of consecutive robots is N/k. This is the minimal value that can be reached. Therefore, clearly, condition **a** guarantees this minimality.

If the robots are not coordinated, then it is possible that two consecutive robots along the cycle, R_i and R_{i+1}, will move in opposite directions. Therefore the distance between them will increase from $\frac{N}{k}$ to $\frac{N}{k}+2$, and by Lemma 2 the ppd in the segments between them will be smaller. If R_i and R_{i+1} move towards one another, then the distance between them will be $\frac{N}{k}-2$ and the ppd in the segments between them will become higher. On the other hand, some pair R_j and R_{j+1} exists such that the distance between them increases, as the total sum of distances between consecutive robots remains N, hence the minimal ppd around the cycle will become smaller.

Therefore the only way of achieving the minimal distance (maximal ppd) is by assuring that condition **a** is satisfied, and maintaining it is achieved by satisfying condition **b**. □

Corollary 4. *In the full-knowledge adversarial model, an optimal patrol algorithm must guarantee that the robots are placed uniformly along the perimeter throughout the execution of the patrol.*

In the zero-knowledge adversarial model, the requirement remains. In this model, the optimal patrol algorithm should maximize the *expected* ppd throughout the perimeter. Since we assume the penetration spot is chosen at random with uniform distribution, the expected ppd is the average ppd in all segments. Recall that the probability that a penetrator will be detected in segment s_i by robot R_j during t time units is ppd_i^j, hence $\mathsf{ppd}_i = \sum_{j=1}^{k} \mathsf{ppd}_i^j$. The expected ppd is, then, $\mathbb{E}(\mathsf{ppd}) = \sum_{i=1}^{N} \sum_{j=1}^{k} \mathsf{ppd}_i^k$. However, $\sum_{j=1}^{k} \mathsf{ppd}_i^k$ might be greater than 1, therefore the overall expected ppd is $\mathbb{E}(\mathsf{ppd}) = \sum_{i=1}^{N} \min\{1, \sum_{j=1}^{k} \mathsf{ppd}_i^k\}$. If $\sum_{j=1}^{k} \mathsf{ppd}_i^k < 1$ is always true, then the location of the robots is irrelevant to the value of $\mathbb{E}(\mathsf{ppd})$.

However, this is not the case - consider for example the case in which the robots are located in adjacent segments. We have shown in Lemma 5 that as the distance between a robot R_i and a segment s_j increases, the probability of reaching it during t time units decreases, i.e., ppd_j^i decreases. In order to maximize the expected ppd, it is necessary to place the robots such that $\forall s_i$, $1 \leq i \leq N$, $\sum_{j=1}^{k} \mathsf{ppd}_i^k \leq 1$. Since the patrol path is circular, decreasing the distance between two robots R_a and R_b, necessarily increases the distance between two robots R_x and R_y. Therefore the optimal placement of the robots is with uniform distance between them, i.e., $d = N/k$. To guarantee that this optimality measure is maintained, the robots must be kept synchronized, i.e., if they choose to switch directions they should do it simultaneously.

Following this chapter, we focus in this work on perimeter patrol in the $\mathcal{DCP}$ movement model, where the robots' patrol algorithm has the following characteristics.

- The robots are placed uniformly around the perimeter with d segments between every two consecutive robots.
- The robots are coordinated in the sense that if they decide to turn around, then they do it simultaneously.
- At each time step, the robots continue straight with a probability of p or turn around with a probability of $1 - p$, and if they turn around they stay in the same segment for τ time units.

Note that under the above assumptions (i.e. the algorithm framework for homogenous robots), the division of the perimeter into sections of d segments creates an equivalent symmetric environment in the sense that in order to calculate the optimal patrol algorithm it is sufficient to consider only *one section* of d segments, and not the entire perimeter of N segments. This is due to the fact that each section is completely equivalent to the other, and remains so throughout the execution.

We divide the goal of finding the *optimal* patrol algorithm into two stages.

1. Calculating the d ppd_i functions for each $1 \leq i \leq d$. This is determined according to the robotic movement model, environment model (perimeter/fence) and sensorial model (perfect/imperfect, local/extended).
2. Optimizing an objective function according to the adversarial model, based on the given ppd_i functions. For example, given a strong adversary that will penetrate through the segment with minimal ppd, the objective function would be to maximize the ppd in the segment with minimal ppd.

The first stage is described in Chapter 4, and the second stage is described in Chapter 5.

Chapter 4

Determining the Probability of Penetration Detection

In order to find the *optimal* patrol algorithm, it is necessary to first determine the probability of penetration detection at each segment s_i (ppd_i), which is a function of p (the probability characterizing the patrol algorithm, as shown in Chapter 3). In this chapter we present various algorithms in order to determine this probability, based on the movement model, the environment model and the sensorial abilities of the robots. Specifically, in Section 4.1 we describe the basic algorithm for determining ppd_i in the perimeter patrol problem, with the various movement models of the robots. We assume that the robots have perfect sensing abilities, i.e., if the adversary lies inside their sensorial range, the robots will surely detect it. Section 4.2 handles the case in which the robots have *imperfect* sensing abilities, where they will detect the adversary only with some probability $p_d \leq 1$. Section 4.3 considers the case of *fence patrol*. It first describes how it is different from the perimeter patrol problem, and then it describes an algorithm for determining ppd_i for both perfect and imperfect sensing abilities of the robots. Last, in Section 4.4 we discuss further enhancements to the sensing abilities of the robots in perimeter patrol.

4.1 Basic algorithm for determining ppd_i in perimeter patrol

In this section we present a polynomial time algorithm for determining ppd_i $\forall 1 \leq i \leq d$ for the basic case of perimeter patrol in which robots have perfect sensing, i.e., a robot will detect the adversary if it is under its sensorial range with a probability of 1.

As stated previously, we need to consider only one section of d segments that lie between two consecutive robots, without loss of generality, R_1 and R_2. We use a Markov chain in order to model the possible states of the system. In this section we describe modeling under the $\mathcal{DCP}$ movement model (the cases of $\mathcal{BMP}$ and $\mathcal{DNCP}$ are similar, and are both described briefly later and illustrated in Figure 4.1).

In order to calculate the probability of detection in each segment along t time cycles, we use the graphic model G illustrated in Figure 4.1. For each segment s_i in the original path, $1 \leq i \leq d-1$, we create two states in G: One for moving in a clockwise direction (s_i^{cw}), and the other for moving in a counterclockwise direction (s_i^{cc}). As mentioned previously, if R_1 or R_2 reach one of the s_i segments within t time units, then the intruder is discovered, i.e., it does not matter if the segment is visited more than once during these t time units. Therefore we consider only the probability of the *first* visit to each segment, and this is done by defining the states s_0 and s'_0 as *absorbing states*. The edges of G are as follows. One outgoing edge from s_i^{cw} to s_i^{cc} exists with a probability of $q = 1 - p$ for turning around, and one outgoing edge to s_{i-1}^{cw} exists with a probability of p for continuing straightforward. Similarly, one outgoing edge from s_i^{cc} to s_i^{cw} exists with a probability q for turning around, and one outgoing edge to s_{i+1}^{cc} exists with a probability of p for continuing straightforward.

If we use the $\mathcal{DNCP}$ model, the chain is similar to the one above, however edges will exist from s_i^{cw} to s_{i+1}^{cc} and from s_i^{cc} to s_{i-1}^{cw} with a probability of q. When we use the $\mathcal{BMP}$ model, the chain is simple: edges exist from s_i to s_{i+1} with a probability of p and from s_i to s_{i-1} with a probability of q (with no related direction). See Figure 4.1 for an illustration of $\mathcal{DNCP}$ and $\mathcal{BMP}$ as a Markov chain.

The straightforward way of finding the probability of arriving at an absorbing vertex is to use a stochastic matrix M, which represents the probability of transition between states. In order to find the probability of absorption

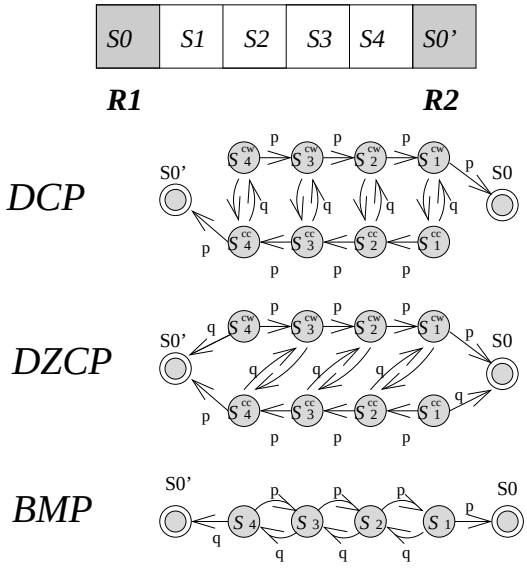


Figure 4.1: Conversion of the initial segments and robot locations for the three possible robotic models: $\mathcal{DNCP}$, $\mathcal{DCP}$ and $\mathcal{BMP}$ into a graphical model.

after t cycles starting from each state s_i, $1 \leq i \leq d-1$, M^t should be computed. However, as t reaches $d = \frac{N}{k}$, it leads to a computational complexity exponential in the input size.

One might consider using the convergence theorem of stochastic matrixes, which states that some t_0 exists such that $\forall t' > t_0$, $M^{t'}$ converges to some matrix $\tilde{M}$. However, the t values we consider are smaller than t_0 in this theorem, therefore it is irrelevant.

Consequently we use the following dynamic-programming inspired rule in order to find the *optimal* solution, yet in polynomial time. We determine the probability of reaching a certain state in time r by the sum of probabilities of reaching s_i from any other state s_j multiplied by the probability of being in state s_j at time $r-1$. Hence in order to compute the probability of reaching absorption state in t time cycles starting from state s_{init}, we initialize s_{init} with the value 1 at $r = 0$, compute the values for $r = 1, \ldots, t$, and extract the probability at the absorption state, s_{abs}. See Algorithm FindFunc for a detailed description of the method.

The time complexity of Algorithm FindFunc is $d \cdot (2d+2) \cdot (t+1)$. Since t is bounded by $d-1$ and $d = N/k$, then the complexity is $\mathcal{O}((\frac{N}{k})^3)$.

Algorithm 1 Algorithm FindFunc(d, t)

for each $s_{init} = s_i \in \{s_1, \ldots, s_{d-1}\}$ **do**
  Create matrix M of size $(2d+2) \times (t+1)$, initialized with 0s
  Set $M_0(s_{init}) \leftarrow 1$
  Complete M gradually using the following rules.
  for each entry $M_r(s_i^{cw})$ **do**
    Set value to $p \cdot M_{r-1}(s_{i+1}^{cw}) + q \cdot M_{r-1}(s_i^{cc})$
  for each entry $M_r(s_i^{cc})$ **do**
    set value to $p \cdot M_{r-1}(s_{i-1}^{cc}) + q \cdot M_{r-1}(s_i^{cw})$
  for absorbing states **do**
    Set entry $M_r(s_{abs}) = M_{r-1}(s_{abs}) + p \cdot [M_{r-1}(s_1^{cw}) + M_{r-1}(s_d^{cc})]$
Report row t of M

4.2 Perimeter patrol with imperfect detection

Uncertainty in the perception of the robots should be taken into consideration in practical multi-robot problems. Therefore we consider the realistic case in which the robots have imperfect sensorial capabilities. In other words, even if the adversary passes through the sensorial range of the robot, it still does not necessarily detect it.

We introduce the ImpDetect model, in which a robot travels through one segment per time cycle along the perimeter while monitoring it, and has imperfect sensing. Denote the probability that an adversary penetrating through a segment s_i while it is monitored by some robot R and R will actually detect it by $p_d \leq 1$.

Note that if $p_d < 1$, revisiting a segment by a robot could be worthwhile - it could increase the probability of detecting the adversary. Therefore the probability of detection in a segment s_i (ppd_i) is *not* equivalent to the probability of *first* arriving at s_i (as illustrated in Section 4.1), but the probability of detecting the adversary during *some* visit y to s_i, $0 \leq y \leq t$. Denote the probability of the y'th visit of some robot to segment s_i by w_i^y. Therefore ppd_i is defined as follows.

$$\mathsf{ppd}_i = w_i^1 p_d + w_i^1(1-p_d) \times \{w_i^2 p_d + w_i^2(1-p_d) \times \{\ldots\{w_i^t \times p_d\}\}\} \quad (4.1)$$

In other words, the probability of detecting the penetration is the proba-

bility that it will be detected in the first visit ($w_i^1 \times p_d$) plus the probability that it will *not* be detected then, but during later stages. This again is the probability that it will be detected during the second visit ($w_i^2 \times p_d$) or at later stages, and so on.

Note that after t time units, $w_i^t = 0$ for all currently unoccupied segments s_i, and if a robot resides in s_i, then w_i^t is precisely $(1-p_d)^t$.

One of the building blocks upon which the optimal patrol algorithms are based, is the assumption that the probability of detection decreases or remains the same as the distance from a robot increases, i.e., it is a monotonic decreasing function. This fact was used in Chapter 3 in proving that in order to maintain an optimal ppd, the robots must be placed uniformly around the perimeter (with a uniform time distance), and maintain this distance by being coordinated. In order to show this here as well, we first prove that the probability of detection monotonically decreases with the distance from the location of the robot.

Lemma 5. *Let* $S = \{s_{-t+\tau}, \ldots, s_{-1}, s_0, s_1, \ldots, s_t\}$ *be a sequence of* $2t$ *segments, where robot* R_a *resides in* s_0 *at time* 0. *Then* $\forall i \geq 0$, $\mathsf{ppd}_i \geq \mathsf{ppd}_{i+1}$, *and* $\forall i \leq 0$, $\mathsf{ppd}_i \geq \mathsf{ppd}_{i-1}$.

Proof. First, assume that $i > 0$ (positive indexes). By Equation 4.1, we need to compare between $w_i^1 p_d + w_i^1(1-p_d) \times \{w_i^2 p_d + w_i^2(1-p_d) \times \{\ldots\{w_i^t \times p_d\}\}\}$ and $w_{i+1}^1 p_d + w_{i+1}^1(1-p_d) \times \{w_{i+1}^2 p_d + w_{i+1}^2(1-p_d) \times \{\ldots\{w_{i+1}^t \times p_d\}\}\}$. It is therefore sufficient to show that $w_i^m \geq w_{i+1}^m$, for all $1 \leq m \leq t$. We prove this by induction on m. As the base case, consider $m = 1$, i.e., we need to show that $w_i^1 \geq w_{i+1}^1$. This is accurately proven in Lemma 1, based on the fact that the movement of the robots is continuous, therefore in order to get to a segment you must visit the segments in between (the formal proof also uses the conditional probability law).

We now assume correctness for $m' < m$, and prove that $w_i^m \geq w_{i+1}^m$. Denote the probability that a robot placed at segment s_i will return to s_i within r time units by $x_i(r)$. In our symmetric environment, for every i and j, $x_i(r) = x_j(r)$. Moreover, $\forall r, x_i(r) \geq x_i(r-1)$. Therefore w_i^m can be described as $\sum_{r+u \leq t} w_i^{m-1}(u) \times x_i(r)$, and similarly $w_{i+1}^m = \sum_{r+u \leq t} w_{i+1}^{m-1}(u) \times x_{i+1}(r)$. By the induction assumption, $w_i^{m-1} \geq w_{i+1}^{m-1}$, and since $x_i(r) = x_{i+1}(r)$, it follows that $w_i^m \geq w_{i+1}^m$, proving the lemma for positive indexes.

The negative indexes are a reflective image of the positive indexes, but with $t - \tau$ time units. Since the induction was proven for all t values, the proof for the negative indexes directly follows. □

The following Theorem follows directly from Lemma 5. The idea behind this is that since the probability of penetration detection decreases as the distance from the robots grow, both minimal ppd and average ppd are maximized if the distance between the robots is as small as possible. Since the patrol path is cyclic, this is achieved only if the distance between every two consecutive robots is uniform, and remains uniform.

Theorem 6. *For both the full knowledge and zero knowledge adversarial models, a patrol algorithm in the* ImpDetect *model is optimal only if it satisfies two conditions: a. The robots are placed uniformly around the perimeter. b. The robots are coordinated in the sense that if they turn around, they do it simultaneously. By assuring these two conditions, the robots preserve a uniform distance between themselves throughout the execution.*

Algorithm for finding ppd_i with imperfect sensorial detection:

We now describe Algorithm FindPPDwImpDetect that finds the probability of penetration detection in each segment (ppd_i). The algorithm computes the probability of all visits to a segment during t time units. This algorithm, similar to Algorithm FindFunc, is inspired by dynamic programming. As stated previously, the main difference between the algorithms is that FindFunc considers only the first visit to a segment, whereas FindPPDwImpDetect considers *all* visits to a segment and the probability of sensorial detection. Figure 4.2 describes a representation of transition between segments as a Markov chain. This is later translated into gradually constructing a table using a dynamic programming-inspired rules, as described in Algorithm FindPPDwImpDetect. The time complexity of the algorithm is $\mathcal{O}(dt)$, which is the time it takes to construct table M. Extracting the polynomial coefficient is done in $\mathcal{O}(1)$ time.

Theorem 7. *Algorithm* FindPPDwImpDetect(d, t, i) *computes* ppd_i.

Proof. In order to prove the theorem, we show that the algorithm correctly computes the probability of the m'th arrival to s_i, for every $1 \leq m \leq t$, i.e., the coefficient of f^m is precisely w_i^m. Since each path is multiplied by f each time it passes through s_i, then inevitably if the path goes through s_i m times, it is a coefficient of f^m. By adding all arrivals to s_0^{cw} and s_0^{cc}, we take into consideration all visits to s_i, hence all paths going through s_i are taken into consideration, and by using Equation 4.1, we accurately generate ppd_i. □

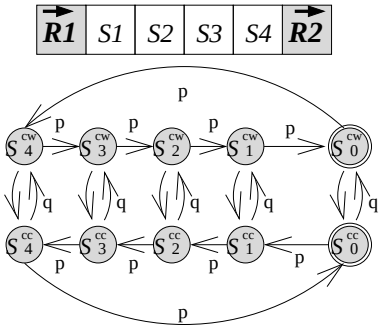


Figure 4.2: Representation of the system as a Markov chain along with state transition. The robots are initially placed at the external segments, heading right. State s_0 represents the segment currently occupied by a robot.

4.3 Fence patrol

In our general work, and specifically in previous sections, we assumed the robots travel around a closed, circular, area. In this section we discuss patrolling along an open polyline, also known as *fence patrol*. First, we will discuss how this patrol is different from perimeter patrol. We will then describe an algorithm for determining ppd_i in fence patrol assuming the robots have perfect sensing capabilities, and finally we will provide an algorithm for robots with imperfect sensing.

4.3.1 Patrolling along a closed polyline vs. an open polyline

In the following, we describe why patrolling along an open polyline is more challenging than patrolling in cyclic environments (closed polyline).

The first reason lies in the fact that the robots are required to go back and forth along a part (or parts) of the open polyline. As a result, the elapsed time between two visits of a robot at each point along this line can be almost twice as long as the elapsed time in a circular setting. In Figure 4.3, we are given two environments: a closed polyline (circle) (a) and an open polyline (b). Note that open polylines b. and c. are equivalent in the sense that each robot travels through one segment per time step, regardless of the shape of the section. Both lines a. and b. are of the same total length l and with the same number of robots (4). In the circular environment, if it takes an

Algorithm 2 FindPPDwImpDetect(d, t, loc)

1: Create matrix M of size $(2d+2) \times (t+1)$, initialized with 0s.
2: Set $M[0, loc^{cw}] \leftarrow 1$.
3: Fill all entries in M gradually using the following rules.
4: **for** $r \leftarrow 1$ to t **do**
5: **for** $i \leftarrow 1$ to d (all other states) **do**
6: For each entry $M[r, s_i^{cw}]$ set value to $p \cdot M[r-1, s^{cw}_{(i+1 \mod d)}] + q \cdot M[r-1, s_i^{cc}]$.
7: For each entry $M[r, s_i^{cc}]$ set value to $p \cdot M[r-1, s^{cc}_{(i-1 \mod d)}] + q \cdot M[r-1, s_i^{cw}]$.
8: **for** s_0^{cw} and s_0^{cc} **do**
9: Set $M[r, s_0^{cw}] \leftarrow f \times \{p \cdot M[r-1, s_1^{cw}] + q \cdot M[r-1, s_0^{cc}]\}$
10: Set $M[r, s_0^{cc}] \leftarrow f \times \{p \cdot M[r-1, s_d^{cc}] + q \cdot M[r-1, s_0^{cw}]\}$
11: $w^i_{loc} \leftarrow$ polynomial coefficients of f^i from sum of $M[r, s_0^{cw}] + M[r, s_0^{cc}]$, for all $0 \le r \le t$, $1 \le i \le t$.
12: Return the result obtained by substituting the w^i_{loc} values in Equation 4.1.

adversary more than $l/4$ time units to penetrate - it will never be able to penetrate even if the robots simply continuously travel with uniform distance between them. However, if the robots travel along an open polyline (b), the maximal time duration between two visits of the robot—even in the best case, is $2l/4 - 2$ [33]. Therefore a weaker adversary that has a penetration time which is almost twice as long as in the circular fence might still be able to penetrate.

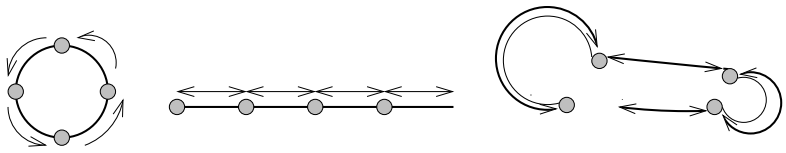


Figure 4.3: Illustration of the difference between patrolling along a line and patrolling along a circle, for different polylines

Another reason for the added complication in analyzing the probability of penetration detection in open polyline environments lies in the asymmetric nature of traveling in the segments along time. In a circular environment, if

the robots are coordinated and switch directions in unison, then the placement of the robots is symmetric in each time unit. Therefore all segments in the same distance from some robot (with respect to its direction) have the same probability of penetration detection. Hence in order to calculate the optimal way of movement (in our case the probability p of turning around), it is sufficient to consider only one section of d segments, and the resulted p is equivalent throughout the execution. In an open polyline environment this is not the case. The probability of penetration detection differs with respect to the current location and direction of the robot. Therefore the algorithm that finds the ppd for each segment, needs to calculate the ppd as a function of p for each segment s_i for each possible initial location of the robot inside the section. Therefore this results in a matrix of size $d \times d$ of the ppd functions (as opposed to a vector of d functions in the circular fence).

4.3.2 Determining ppd_i in an open polyline

The *best* patrol algorithm for a team of robots in an adversarial environment requires to find an algorithm that will maximize some function of the probability of penetration detection (ppd). Therefore in this section we describe Algorithm FindFencePPD that finds the ppd in the segments.

The ppd in a segment s_i is determined by the probability of the *first* arrival to s_i during t time units. This is due to the fact that we assume, at this stage, that if the robot arrives at a segment that is currently occupied by the adversary, it will detect it. Consequently, whether the segment is visited more than once during t time units is irrelevant. For simplicity, we discuss the case in which $\tau = 1$.

Similar to the fence patrol case (Section 4.1), we describe the system as a Markov chain (see Figure 4.4). Since the robots have directionality associated with their movement, we create two states for each segment: the first for traveling in a segment in the clockwise direction, and the second for traveling in the counterclockwise direction. The probability of turning around at the end of each section is 1, otherwise the robot will continue straight with a probability of p, and will turn around with a probability of $q = 1 - p$.

The algorithm is inspired by dynamic programming, and uses the state transition rules described in Figure 4.4 in order to gradually fill in a matrix M, until it reaches the state of the system after t time units (last line). The algorithm runs as follows. In order to extract the *first* visit to a segment,

each segment s_i is associated with a "phantom" variable f_i. Whenever an element goes through s_i's location in the matrix, it is multiplied by f_i. At the end, all visits to the segment are added, and all phantom variables are substituted with 1 *except* for f_i. Therefore the polynomial coefficient of f_i^t is the probability of the t'th visit to s_i, the coefficient of f_i^{t-1} represents the probability of the $t-1$ visit, and so on. Hence ppd_i in this case is the polynomial visit of f_i^1, i.e., the probability of the first visit to s_i.

Recall that $d = N/k$. The returned value from Procedure FindFencePPD is a matrix of size $d \times d$, where each row i contains d functions representing the ppd in each segment, given that the robot is currently in segment s_i.

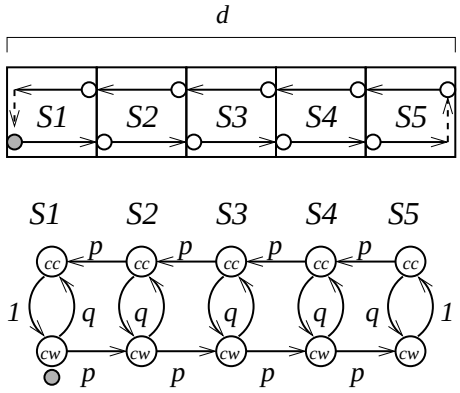


Figure 4.4: Description of the system as a Markov chain, as base for the FindFencePPD algorithm.

Time complexity: The time complexity of Procedure FindFencePPD is $\mathcal{O}(d^2t)$, since we fill in the matrix M d times, where each time it takes $d \cdot t$ time to fill it. Hence the total time complexity is $\mathcal{O}(d \cdot d \cdot t) = \mathcal{O}(d^2t)$.

4.3.3 Fence patrol with imperfect detection

Considering that patrolling robots' imperfect sensing is essential to adapt the system to realistic robotic environments, a solution for imperfect sensing in perimeter patrol was presented in Section 4.2. In this section we provide a solution for handling these sensing abilities in fence patrol.

As previously mentioned the probability that the adversary will be detected by a robot currently visiting the segment in which it resides is denoted

Algorithm 3 Procedure FindFencePPD(d, t)

for $loc \leftarrow 1$ to d **do**
  Create the matrix Res of size $d \times d$, initialized with 0s.
  Create the matrix M of size $2d \times (t+1)$ initialized with 0s.
  Set $M[1, s_{loc}^{cw}] \leftarrow f_{loc}$
  Fill all entries in M gradually using the following rules.
  for $r \leftarrow 2$ to t **do**
    Set $M[r, s_1^{cw}] \leftarrow f_1 \times M[r-1, s_1^{cc}]$
    Set $M[r, s_d^{cc}] \leftarrow f_1 \times M[r-1, s_d^{cw}]$
    for all other entries in row r **do**
      Set $M[r, s_i^{cw}] \leftarrow v_i \times \{p \cdot M[r-1, s_{i+1}^{cw}] + q \cdot M[r-1, s_i^{cc}]\}$.
      Set $M[r, s_i^{cc}] \leftarrow v_i \times \{p \cdot M[r-1, s_{i-1}^{cc}] + q \cdot M[r-1, s_i^{cw}]\}$.
  Create the vector V of size d
  for $j \leftarrow 1$ to d **do**
    $V[j] \leftarrow \sum_{i=1}^{t} M[i, s_j^{cw}] + M[i, s_j^{cc}]$
    For each entry i in V, substitute all f_k, $k \neq j$ with 1.
    Set $Res[loc, j] \leftarrow$ polynomial coefficient of f_j in $V[j]$.
  Set $Res_k \leftarrow VM$.
Return Res.

$p_d \leq 1$. Therefore the probability of detection in a segment s_i (ppd_i) is *not* equivalent to the probability of *first* arriving at s_i, but the probability of detecting the adversary during some visit m to s_i, assuming it was not previously detected. The probability of the y'th visit of some robot to segment s_i is denoted w_i^y. Therefore ppd_i is defined as in Equation 4.1. Note that the difference of determining ppd_i in the fence patrol model compared to the ImpDetect model (which considers the perimeter patrol) is in determining w_i^y, as described in Algorithm ComputeProbPPD.

Algorithm 4 Procedure ComputeProbPPD(d, t)

1: Run Procedure FindFencePPD(d, t) while returning the entire polynomial, i.e., skipping line 16 of the procedure.
2: **for** $j \leftarrow 1$ to d **do**
3: **for** $i \leftarrow 1$ to d **do**
4: $w_i^y \leftarrow$ polynomial coefficient of f_i^y.
5: $PRes[j, i] \leftarrow$ substitution of $w_i^y, \forall y$ in Equation 4.1.

Theorem 8. *For each segment s_i, Procedure* FindFencePPD *computes* ppd_i.

Proof. In order to prove the theorem, we need to show that the algorithm correctly computes the probability of the m'th arrival to s_i, for every $1 \leq m \leq t$, i.e., the coefficient of f^m is exactly w_i^m. Since each path is multiplied by f each time it passes through s_i, then essentially if the path went through s_i m times, it is a coefficient of f^m. By adding all arrivals to s_0^{cw} and s_0^{cc}, we take into consideration all visits to s_i, hence all paths going through s_i are taken into consideration, and by using Equation 4.1, we accurately generate ppd_i. □

As in FindFencePPD, the returned value is a matrix of size $d \times d$, where each row i contains d functions representing the ppd in each segment, given that the robot is currently in segment i.

4.4 Improving sensing capabilities in perimeter patrol

In this section we present further enhancements by considering various sensing capabilities of the robots. Specifically, we first consider the case in which

a robot can sense beyond its currently visited segment. We then offer a solution to the case in which the robot can sense beyond its current position, yet its sensing capabilities are not perfect, and change as a function of the distance from its current position.

4.4.1 Extending sensing range

In this section we consider the LRange model, in which the sensorial range of a robot exceeds the section in which it currently resides. Use L to denote the number of segments the robot senses beyond the segment it currently occupies (see Figure 4.5). If $L > 0$, we refer to the L segments as *shaded segments*. Note that the location of the shaded segments depends on the direction of the robot shading them, and they are always in the direction the robot is facing.

A trivial solution to dealing with such a situation is to enlarge the size of the segment, and thus enlarge the length of the time unit used as base for the system, such that it will force L to be 0. However, in this case we lose accuracy of the analysis of the system, as the length of the time cycle should be as small as possible to also suit the velocity of the robots and the value of t.

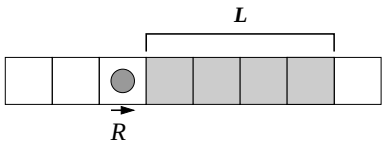


Figure 4.5: An illustration of L segments shaded by robot R. In this case R is facing right, therefore the shaded segments are to its right.

In general, the values of t that can be handled by the system are bounded by its relation to d (the distance between every two robots along the path) - see Chapter 3. If $L > 0$, this changes. Specifically, if $L = 0$, then the possible values of t considered are $\lceil d/2 \rceil + \tau \leq t \leq d - 1$. However, if $L > 0$, then it is possible to handle even smaller values of t, i.e., even if the penetration time of the adversary is short. Formally, the possible values of t are given in the following equation.

$$\lceil d/2 \rceil + \tau - L \leq t \leq d - L$$

If t is smaller than $\lceil d/2 \rceil + \tau - L$, then an adversary with full knowledge will manage to penetrate with a probability of 1, i.e., there is a segment which is unreachable within t time units. On the other hand, if t is greater than $d-L$, then a simple deterministic patrol algorithm will detect all penetrations with a probability of 1. We assume that during the τ time units the robot turns around, it can sense only its current segment.

Algorithm for finding ppd_i with shaded segments:

For each segment s_i, ppd_i is determined by the probability that some robot will visit this segment *plus* the probability that this segment will be shaded by some robot. We use a dynamic-programming inspired rule, similar to the one described in Section 4.1, yet we expand it to also include the probability that it will be shaded by some robot. The main idea is that in each transition phase, the algorithm checks whether the state shades on an absorbing state, i.e., whether the robot in its current location and direction shades the given segment (the distance from it is smaller than L). See Algorithm 5 for a full description of the algorithm. The time complexity of the algorithm is $\mathcal{O}(dt)$, which is the time it takes to fill in the entire table.

Algorithm 5 $\mathsf{FindPPDwShade}(d, t, loc, L)$

Create matrix M of size $(2d+2) \times (t+1)$, initialized with 0s.
Set $M[0, loc^{cw}] \leftarrow 1$.
Fill all entries in M gradually using the following rules.
for $r \leftarrow 1$ to t **do**
  for each entry $M[r, s_i^{cw}]$ **do**
    Set $v \leftarrow p \cdot M[r-1, s_{i+1 \mod d}^{cw}] + q \cdot M[r-1, s_i^{cc}]$
    Set $M[r, s_i^{cw}] \leftarrow v$
    if $i + L \geq d$ **then**
      Set $M[r, s_{abs}] \leftarrow v$
  for each entry $M[r, s_i^{cc}]$ **do**
    Set $v \leftarrow p \cdot M[r-1, s_{i+1 \mod d}^{cw}] + q \cdot M[r-1, s_i^{cc}]$.
    Set $M[r, s_i^{cc}] \leftarrow v$
    if $i - L \leq 0$ **then**
      Set $M[r, s_{abs}] \leftarrow v$
  for absorbing state $M_r(s_{abs})$ **do**
    Set $M[r, s_{abs}] \leftarrow M_{r-1}(s_{abs}) + p \cdot [M_{r-1}(s_1^{cw}) + M_{r-1}(s_d^{cc})]$
Return $M[t, s_{abs}]$

4.4.2 Extending the sensorial range along with imperfect detection

In many cases, the actual sensorial capabilities of the robot are composed of the two characteristics described in the previous sections, i.e., the robot can sense beyond its current segment, however the sensing ability is imperfect. Therefore in this section we introduce the ImpDetLRange sensorial model, which is a combination of the LRange and the ImpDetect models. Here the robot can sense L segments beyond its current segment, yet the p_d in each segment varies and is not necessarily 1. We therefore describe an algorithm that deals with the most realistic form of sensorial capabilities [29]: imperfect, long range sensing.

The information regarding the sensorial capabilities of the robots includes two parameters. The first describes the quantity of the sensing ability, i.e., the number of segments that exceeds the current segment in which robot resides, for which it has *some* sensing abilities, denoted by L. The second parameter describes the quality of sensing in all segments the robot can sense. This is given in the form of a vector $V_S = \{v_0, v_1, \ldots, v_L\}$, where v_i is the probability that the robot residing in s_0 will detect a penetration that occurs in segment s_i. We assume that the values in V_S decrease monotonically, i.e., as i increases, v_i decreases or remains the same.

In the ImpDetLRange model, the probability of penetration detection is more complex, and also has to take into consideration the possibility of being in the sensorial range of some robot and the probability of being detected there. Denote the probability that s_i is at a distance of $j <= L$ from some robot, i.e., within its sensorial range, for the j'th time by $w_i^j(e)$. Denote the probability that the adversary in s_i will not be detected at all by $\overline{\mathsf{ppd}_i}$. The probability that the adversary will be detected is actually the complementary of the probability that it will not be detected. Therefore ppd_i is defined as follows.

$$\mathsf{ppd}_i = 1 - \overline{\mathsf{ppd}_i} = 1 - \prod_{j=1}^{t} \prod_{e=1}^{L} \{w_i^j(e) \cdot (1 - v_e)\} \tag{4.2}$$

In other words, the probability of penetration detection is the complementary of the probability that the adversary will *not be detected at all* during the t time units. This is the probability that it is not detected at any possible occurrence in any possible range (corresponding to a probability of

detection) during those time units. The overall number of components is, therefore, $L \times t$.

Algorithm for finding ppd_i with an extended-range and imperfect detection:

The algorithm used to find ppd_i if we allow an extended range ($L > 0$) and imperfect detection with changing probabilities of detection as a function of the distance from the robot is composed of two stages. In the first stage, we need to find the probability of being shaded with a distance of $1 \leq e \leq L$ from the robot for the j'th time, $1 \leq j \leq t$. This provides us with all $w_i^j(e)$ values. We then substitute all the acquired values in Equation 4.2. The full description of the algorithm is presented in Algorithm FindComplexP below. Note that in Algorithm FindPPDwImpDetect we had one object f used to identify the number of visits to the segment. In this case, since we have to consider all visits of all possible distances that are less or equal to L (shaded segments), we use $L + 1$ objects, $f_0, \ldots, f_L$. The time complexity of the algorithm is $\mathcal{O}(dt + Lt)$ - the time to construct the M table plus the time to extract all polynomial coefficients (respectively). Since $L < d$, this is again $\mathcal{O}(dt)$.

Algorithm 6 FindComplexP$(d, t, loc, L, V_S = \{v_0, \dots v_L\})$

Create matrix M of size $(2d+2) \times (t+1)$, initialized with 0s.
Set $M[0, loc^{cw}] \leftarrow 1$.
Set $Res \leftarrow 0$
Fill all entries in M gradually using the following rules.
for $r \leftarrow 1$ to t **do**
  for each entry $M[r, s_i^{cw}]$ **do**
    Set $u \leftarrow p \cdot M[r-1, s_{i+1 \bmod d}^{cw}] + q \cdot M[r-1, s_i^{cc}]$
    if $i + L \geq d$ **then**
      $u \leftarrow u \times f_{d-i}$
      $Res \leftarrow Res + u$
    Set $M[r, s_i^{cw}] \leftarrow u$
  for each entry $M[r, s_i^{cc}]$ **do**
    Set $u \leftarrow p \cdot M[r-1, s_{i+1 \bmod d}^{cw}] + q \cdot M[r-1, s_i^{cc}]$.
    if $i - L \leq 0$ **then**
      $u \leftarrow u \times f_i$
      $Res \leftarrow Res + u$
    Set $M[r, s_i^{cc}] \leftarrow u$
$w_i^j(e) \leftarrow$ polynomial coefficient of f_e^j of Res, for all $1 \leq j \leq t$, $0 \leq e \leq L$ (while substituting all other $f_{e'}^j$, $e' \neq e$ in the equation).
Return the result obtained by substituting the $w_i^j(e)$ values in Equation 4.2.

Chapter 5

Patrol in Different Adversarial Models

Finding the optimal patrol algorithm depends heavily on the adversarial model the system works under. Specifically, the optimality depends on the knowledge obtained by the adversary on the patrolling robots—the patrol algorithm (characterized by the probability p) and the robots' location.

If the adversary has full knowledge of the patrol scheme, then it will use this information in order to choose a penetration spot such that it will less likely be detected by the patrolling robots. On the other hand, if the adversary has not obtained any knowledge on the patrol scheme, it is assumed that it will choose its penetration spot at random with uniform distribution between the currently unoccupied segments. In Sections 5.1 and 5.2 we provide theoretical results for these two extreme adversarial models—full and zero knowledge adversaries (respectively), showing the optimal patrol algorithm in both cases. Then in Section 5.3 we discuss the case in which the adversary's knowledge is somewhere along the scale of knowledge - between zero and full. We provide two heuristic algorithms for this case, and discuss possible theoretical implications of partial knowledge of the adversary on the choice of the optimal patrol algorithm.

5.1 Handling a full-knowledge adversary

In cases in which the robots face a strong adversary, that has full knowledge of the patrol algorithm, it is assumed that the adversary will take advantage

of this knowledge to find the weakest spot of the patrol, i.e., the segment with minimal probability of penetration detection. Therefore the optimal patrol algorithm to handle such an adversary is the one that maximizes the minimal ppd throughout the perimeter. Hence we need to find the optimal p, p_{opt}, such that the minimal ppd throughout the perimeter is maximized. Formally,

$$p_{opt} = \underset{0 \le p \le 1}{\operatorname{argmax}} \{ \min_{1 \le i \le d-1} f_i(p) \}$$

Since our environment is symmetric, we do not need to consider the entire patrol path, but only a section of d segments between two consecutive robots. The input in this procedure is the set of d ppd_i functions that were calculated in the previous chapter (Chapter 4). It therefore works in all sensorial models of the robots.

We describe our solution for the perimeter patrol case and then provide a generalization for fence patrol.

5.1.1 Finding the maximin point

After establishing d equations representing the probability of detection in each segment, we must find the p value that maximizes the minimal possible value in each segment, where $p \in [0, 1]$. Denote these equations by $f_i(p)$, $1 \le i \le d-1$. The maximal minimal value is the maximal value that lies inside the intersection of all integrals of f_i.

Observing the problem geometrically, consider a vertical sweep line that sweeps the section $[0, 1]$ and intersects with all d curves. It seeks the point p in which the minimal intersection point between the sweep line and the curves, $f^*(p)$, is maximal. This p is the maximin point. Since the segment $[0, 1]$ and the functions $f_1, \ldots, f_{d-1}$ are continuous, this sweep line solution cannot be implemented. We observe that a maximin point is actually the maximal point that lies inside the integral of all curves. We prove in the following lemma that this point is either an intersection point of two curves, or a local maxima of one curve (see Figure 5.1). See Figure 7 for the formal description of Algorithm FindP.

In the following, we prove that Algorithm FindP finds point p such that the maximin property is satisfied.

Lemma 9. *A point p yields a maximin value $f^*(p)$ if the following two properties are satisfied.*

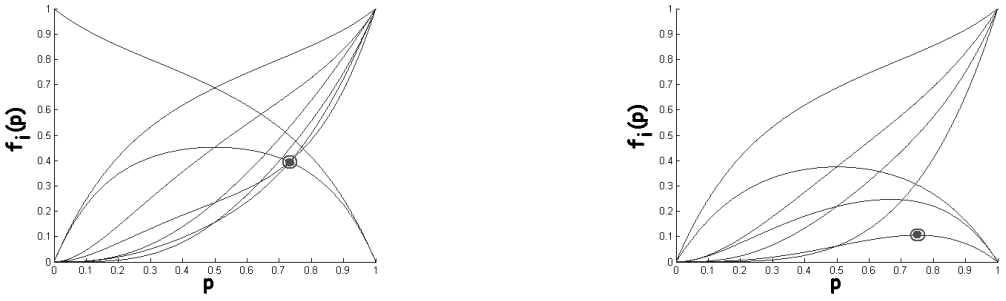


Figure 5.1: An illustration of two possible maximin points. On the left, the point is created by the intersection of two curves, and on in the right it is the local maxima of the lowest curve.

a. $f^*(p) \leq f_i(p)\ \forall 1 \leq i \leq d-1$.
b. *One of the two following conditions holds:* $f^*(p)$ *is an intersection of two curves (or more),* $f_i(p)$ *and* $f_j(p)$ or *a local maxima of curve* $f_k(p)$.

Proof. Property **a.** is derived from the definition of a maximin point. Therefore we are looking for the maximal point that satisfies property a. We must still show that this point, $f^*(p)$, is obtained by either an intersection of two or more curves or is a local maxima. Clearly, a maximal point of an integral is found on the border of the integral (the curve itself). The area which is in the intersections of all curves lies beneath parts of curves, $f_{i_1}, \ldots, f_{i_m}$, such that f_{i_j} is the minimal curve in the section $[l^j, r^j]$ and $\bigcup_{j=1}^{m}[l^j, r^j] = [0, 1]$. By finding the maximal point in each section $f^j_{max} = \max\{f(x), x \in [l^j, r^j]\}$, and choosing the maximal between them, i.e., $\max\{f^j_{max}, 1 \leq j \leq m\}$, we obtain $f^*(p)$. In each section $[l^j, r^j]$ the maximal point can be either inside the section or on the borders of the section. The former case is precisely a local maxima of f_{i_j}. The latter is the intersection point of two curves $f_{i_{j-1}}, f_{i_j}$ or $f_{i_j}, f_{i_{j+1}}$. □

Lemma 10. *A point* p *exists yielding a maximin value* $f^*(p) > 0$.

Proof. In order to prove the lemma, we need to show that the intersection of all integrals $f_1, \ldots, f_{d-1}$ in the x section $[0, 1]$, and the y section $(0, 1]$ is not empty. It suffices to show that for every f_i, $f_i(x) > 0, 0 < x < 1$.

Each function $f_i, 1 \leq i \leq d-1$ represents the ppd in a segment s_i between two robots. From our requirement that $t \geq \lfloor \frac{d}{2} \rfloor$, it follows that in all models we consider, for $0 < p < 1$ the ppd $\neq 0$. Note that if $p = 0$ or $p = 1$, then ppd

is either 0 or 1, but this does not contradict the fact that we have a point guaranteeing $f^*(p) > 0$. □

Algorithm FindP finds this point by scanning all possible points satisfying the conditions given in Lemma 9, and reporting the x-value (corresponding to the p value) with a y-value dominated by all f_i. The input to the algorithm is a vector of functions f_i, $1 \leq i \leq d-1$ and the value t. The time complexity of Algorithm FindP is the complexity of Algorithm FindFunc, $\mathcal{O}((\frac{N}{k})^3)$ plus $\mathcal{O}(d^3) = \mathcal{O}((\frac{N}{k})^3)$ (the algorithm itself), i.e., jointly $\mathcal{O}((\frac{N}{k})^3)$.

Algorithm 7 Algorithm FindP(d, t)

```
F ← Algorithm FindFunc(d, t).
Set p_opt ← 0.
for F_pivot ← F_{1,...,d−1} do
    Compute local maxima (p_max, F_pivot(p_max)) of F_pivot in the range (0, 1).
    for each F_i, 1 ≤ i ≤ d − 1 do
        Compute intersection point p_i of F_i and F_pivot in the range (0, 1).
        if F_pivot(p_i) > F_pivot(p_max) and F_pivot(p_i) ≤ F_k(p_i)∀k then
            p_opt ← p_i.
        if F_pivot(p_max) > F_pivot(p_i) and F_pivot(p_i) ≤ F_k(p_i)∀k then
            p_opt ← p_max.
Return (p_max, F_pivot(p_max)).
```

Theorem 11. *Algorithm* FindP(F, t) *finds point* p *yielding the maximin value of* ppd.

Proof. Algorithm FindP checks all intersection points between the pair of curves, and the points of local maxima of the curves. It then checks the dominance of these points, i.e., whether in the location these points have a lower value compared to all other curves, and picks the maximal of them. Therefore, if such a point is found, by Lemma 9, this point is precisely the maximin point. Moreover, by Lemma 10 this point exists. □

A note on full-knowledge adversary in fence patrol:

In the case of fence patrolling, the ppd value depends on the current location of the robot. Consequently, the optimal p value characterizing the patrol of the robots is different for each segment s_i, where $1 \leq i \leq d$. Note

that there could be different optimal p values with respect to both location and *orientation* of the robot ($2d$ values). However, it is sufficient to calculate the ppd values only d times - only for one direction, as the other direction is a reflective image of the first.

In order to find the maximin point for the fence patrolling case, we use algorithm MaximinFence, which finds the value p such that the minimal ppd is maximized, using Algorithm FindP that computes this point by finding the maximal point in the integral intersection of all curves (ppd_i). The complete description of the algorithm is shown in Algorithm 8.

Algorithm 8 Procedure MaximinFence(d, t)

1: $M \leftarrow$ FindFencePPD(d, t)
2: **for** $i \leftarrow 1$ to d **do**
3: $\quad OpP[i] \leftarrow$ FindP(d, t) with additional given input $M[i]$ as a vector of ppd functions.
4: Return OpP

5.2 Handling a zero-knowledge adversary

In this section we consider the case in which the robots face a weak adversary, that has no knowledge of the robots' patrol algorithm. The only information it has, and upon which it decides where to penetrate, is the current location of the robots. We assume that in this case, the adversary will choose its penetration spot at random with a uniform distribution of the currently unoccupied segments. The patrol algorithm should, therefore, maximize the *expected* ppd along the perimeter, i.e., the average ppd. We show the surprising result that in perimeter patrol, the simple deterministic algorithm is optimal. However in fence patrol, this is not true.

5.2.1 Perimeter patrol

Next, we show that surprisingly, the optimal patrol scheme for handling a zero-knowledge adversary around a perimeter is the deterministic patrol algorithm ($p = 1$), for all $\tau \geq 1$.

Denote the ppd in segment s_j by robot R_0 after switching its direction r times by $\mathsf{ppd}_j^0(r)$.

Lemma 12. *Consider a sequence of $2d$ segments with one robot R_0 in the mid segment at time 0. If the robots switch directions $r \geq 1$ times during t cycles of execution, then $\sum_{j=1}^{2d} \mathsf{ppd}_j^0(r) < t$ for every $\tau \geq 1$.*

Proof. We prove, by induction on r, that for every $r \geq 1$, $\sum_{j=1}^{2d} \mathsf{ppd}_j^0(r) < \sum_{j=1}^{2d} \mathsf{ppd}_j^0(r-1)$. Note that the sum of ppd for $p = 1$ ($r = 0$) is exactly t, hence by proving the induction we prove the lemma.

As the base case, consider $r = 1$ and $\tau = 1$. Note that since $r > 1$ then inevitably $p < 1$. During t cycles, R_0 can visit and monitor at most t segments. Therefore we should consider $t - 1$ segments from both sides of R_0 (one cycle is "wasted" on turning around). We are interested only in the probability of *first* visit at a segment in order to determine the ppd in that segment. Without loss of generality, we assume R_0 heads towards the right.

In order to prove the lemma, it is enough to show that the sum of the expected ppd if $p < 1$ is less than the sum of ppd if $p = 1$, which is t. Thus we check the addition of ppd to the segments to the left of R_0 if $p < 1$, and compare this addition to the reduction of ppd to the segments to the right of R_0 (from 1 in each segment if $p = 1$). Denote the segment in which R_0 initially resides by s_0, the segments to its right in ascending order $(s_1, \ldots, s_t)$ and the segments to its left by descending order $(s_{-1}, s_{-2}, \ldots, s_{-t+1})$. Firstly, the biggest decrease factor is to segment s_t, from 1 to 0. Next, the ppd in each segment s_i, $1 \leq i \leq t-1$, to the right of R_0 decreases from 1 to p^i. Therefore the sum of reduction is $1 + (1-p) + (1-p^2) + \ldots + (1-p^{t-1}) = t - \sum_{i=1}^{t-1} p^i$. On the other hand, the addition of ppd to the segments to the left of R_0 is as follows. The ppd in segment s_{-1} is $(1-p)p + p(1-p)pp + pp(1-p)pppp + \ldots + p^{t/2-1}(1-p)p^{t/2}$. Similarly, the ppd in segment s_{-2} is $(1-p)pp + p(1-p)ppp + \ldots + p^{t/2-3}(1-p)p^{t/2+2}$. Generally, the sum of all ppd in the segments to the left of R_0 is $(1-p)p + (1-p)p^2 + 2(1-p)p^3 + \ldots + \frac{t}{2}(1-p)p^{t-1}$ $= (1-p)[p + p^2 + 2p^3 + 2p^4 + 3p^5 + 3p^6 + \ldots + t/2p^{t-2} + t/2p^{t-1}$. For every $t \geq 2$, $t - \sum_{i=1}^{t-1} p^i$ is greater than the above expression (for $t = 1$, $r = 1$, this is straightforward, as the ppd in all segments except s_0 is 0).

In order to prove the lemma for a general r, we divide the sequence into two: the sequence to the right of R_0 and to the left of R_0. For every $1 < j \leq r$, let $\sum_{i=-t+j+1}^{-1} \mathsf{ppd}_i^0 = \delta(j)$, $\mathsf{ppd}_{-t+j}^0 = \delta'(j)$, $\sum_{i=1}^{t-j} \mathsf{ppd}_i^0 = \alpha(j)$ and $\mathsf{ppd}_{t-j}^0 = \alpha'(j)$ (see Figure 5.2).

We now assume correctness for $r' < r$, i.e., if R_0 switches directions $r' < r$ times during the execution then $\sum_{l=1}^{2d} \mathsf{ppd}_l(R_0) < t$, and prove that this holds also for $r' = r$. We prove this for $\tau = 1$.

The sum of ppd_i^0 for $r-1$ number of direction switches is $\delta(r-1) + \delta'(r-1) + \alpha(r-1) + \alpha'(r-1)$. For r switches, since the robots spend an extra time cycle to turn around, the two extreme segments with $\mathsf{ppd} > 0$ are now unreachable, hence in this case $\delta'(r-1)$ and $\alpha'(r-1)$ no longer exist. Now, $\delta(r) + \delta'(r)$ is similar to changing the initial direction of the robot (by multiplying by $1-p$), and precisely obtaining $\alpha(r-1)$, hence $\delta(r) + \delta'(r) < (1-p)\alpha(r-1)$. Similarly, $\alpha(r) + \alpha'(r) < (1-p)\delta(r-1)$. Altogether, $\sum_{l=1}^{2d} \mathsf{ppd}_l^0(r) = \delta(r) + \delta'(r) + \alpha(r) + \alpha'(r) < (1-p)\alpha(r-1) + (1-p)\delta(r-1)$ and since $(1-p) < 1$ this is smaller than $\sum_{l=1}^{2d} \mathsf{ppd}_l^0(r-1)$. By the induction assumption, this is smaller than t.

The proof follows directly for $\tau > 1$, as the number of segments that become unreachable increase from 1 to τ for each direction switch, while the probability of penetration detection in other segments remains the same. Therefore essentially the sum of ppd after $r \geq 1$ direction switches with cost τ is now considerably smaller than the sum of ppd after r that costs only one extra time cycle, which is less than t, hence with cost τ for each switch this also holds. □

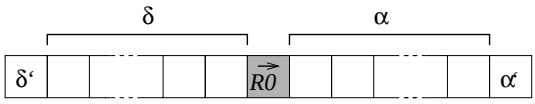


Figure 5.2: Illustration of the proof of Lemma 12.

Corollary 13. *Assume the adversary chooses its penetration segment at random with uniform distribution. Also, assume the robots switch their direction $r > 0$ times during the execution where each direction switch takes τ time units. Therefore the expected* ppd *throughout the perimeter is less than t/d.*

This is because the expected ppd throughout the perimeter is $1/N \sum_{j=i}^{k} \sum_{i=1}^{N} \mathsf{ppd}_i(R_j) < 1/N \sum_{j=i}^{k} t = kt/N = t/d$.

Theorem 14. *The expected* ppd *throughout the perimeter assuming uniform adversary is maximal if the value p characterizing the patrol of the robots equals 1, i.e., the patrol is deterministic for every $\tau \geq 1$.*

Proof. If $p = 1$, then each robot R_j, $1 \leq j \leq k$, assures $\mathsf{ppd}(R_j) = 1$ in exactly t segments, hence the expected ppd throughout the perimeter is

$1/N \sum_j = 1^k t = kt/N = t/d$. Following Corollary 13, every patrol scheme that causes the robots to switch direction once or more, i.e., $p < 1$ has an expected ppd less than t/d. Therefore the deterministic patrol guarantees the maximal expected ppd. □

5.2.2 Perimeter patrol with imperfect sensing

The main idea in the optimality proof of the deterministic algorithm when handling a zero-knowledge adversary, is that it is more beneficial to the robots to visit more segments, since doing so will increase the total ppd. One might draw the conclusion that if the robots have imperfect sensorial capabilities, this argument will not hold since revisiting a segment does have added value. However, we prove the surprising result that even if $p_d < 1$, it is still best to patrol deterministically around the perimeter if the adversary chooses its penetration spot at random. Moreover, we strengthen our result by showing that even if the robot makes a post analysis of its decision to go straight or turn around, it will still decide to keep going straight

Lemma 15. *Assume the adversary picks its penetration spot at random with a uniform distribution between the d unoccupied segments between two robots. Therefore the gain to the robots' probability of penetration detection from revisiting a segment is smaller than the gain from initially visiting a new segment, for every $p_d > 0$.*

Proof. The gain from revisiting a segment, denoted by G_r is the probability that the robot will not detect the adversary during its first visit multiplied by the probability that the adversary indeed will penetrate through that segment. Formally, $G_r = \frac{1}{d} \times (1 - p_d) \times p_d$. On the other hand, the gain from initially visiting a new segment, denoted by G_i is $\frac{1}{d} \times p_d$. Since $1 - p_d < 1$, it follows that $G_r < G_i$. □

Theorem 16. *In the* ImpDetect *model, the deterministic algorithm maximizes the expected* ppd *throughout the perimeter for all $p_d \leq 1$ if the adversary chooses its penetration spot at random with a uniform distribution.*

Proof. Theorem 14 shows that the expected probability of penetration detection is maximal if the robot travels deterministically for $p_d = 1$. The ppd_i in that case was the probability of the *first* visit to a segment s_i. Moreover,

by Lemma 15, a robot adds more to the probability of penetration detection by visiting a *new* segment, rather than revisiting a segment that was already visited once or more. Therefore by adding these two results it follows that the expected ppd is maximized, also when considering as many visits as possible to a segment, when the patrol algorithm is deterministic. □

We strengthen this result by showing that it is beneficial for the robot to keep visiting new segments if the adversary chooses its penetration spot randomly with a uniform distribution (with a probability of $1/d$) even if the robot calculates its benefit post factum, i.e., after visiting a segment. Denote the probability that the adversary will penetrate through segment s_i by PN_i, and the probability that the robot will visit s_i without detecting the adversary by ND_i. Therefore, by the conditional probability law, if $ND_i > 0$, $P(PN_i \mid ND_i) =$

$$\frac{PN_i \bigcap ND_i}{NODEC_i} = \frac{1/d(1-p_d)}{(d-1)/d + 1/d(1-p_d)} = \frac{1-p_d}{d-p_d}$$

On the other hand, the probability that the adversary will choose to penetrate through s_{i+1} knowing that the robot did not detect it in segment s_i is

$$\frac{1 - \frac{1-p_d}{d-p_d}}{d-1} = \frac{1}{d-p_d} > \frac{1-p_d}{d-p_d}$$

In other words, the probability of revealing new information in visiting a new segment is greater than the probability of revealing new information from revisiting a segment that was already visited at least once, even after knowing that the adversary was not caught in the revisited segment. The intuition is that by visiting a new segment, the probability of penetration detection grows by p_d, where if a robot revisits a segment, it carries along with it the probability of arriving there again, multiplied by p_d. Since the probability of arriving again is smaller than 1, the gain from revisiting a segment is smaller.

Another interesting result in this adversarial model is seen in the LRange model, where the optimality of the deterministic algorithm is no longer absolute. The logic behind this statement is that if the cost of turning around (in number of cycles) is smaller than the profit of what the robots gain (number of new visible segments), then it might be worthwhile to turn around. Therefore it is possible to prove optimality of the deterministic algorithm for maximizing the expected ppd only if $L \leq \tau$.

Aa an example to demonstrate the fact that the deterministic algorithm is no longer optimal if $L > \tau$, consider the case in which $L = 2$, $d = 5$ and $\tau = 1$. The maximal expected ppd is 0.85 and it is obtained for $p = 0.5$, whereas if $p = 1$ (deterministic algorithm) the expected ppd is only 0.8.

Theorem 17. *In the* LRange *model, the deterministic algorithm guarantees the maximal expected* ppd *for random-uniform adversary if* $L \leq \tau$.

The proof of this theorem resembles the proof of Theorem 14.

5.2.3 Fence patrol

In circular environments, it was proven that under this adversarial model, the optimal patrol algorithm is the deterministic algorithm. However, if the robot travels along a line - this is no longer the case. We show this by means of a counter example. The logic behind this fact is that in some segments, it is worthwhile to switch the direction of the robot before reaching the end of the section, and thus cover more segments.

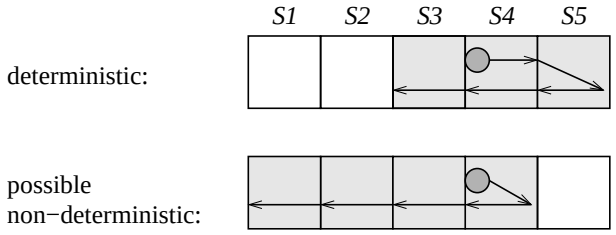


Figure 5.3: An illustration of a case in which the maximal expected ppd is obtained by a non deterministic algorithm. Each arrow represents a movement in one time cycle.

Consider the case in which $d = 5$ and $t = 5$. If the robot is placed in segment s_4, then the ppd values are as follows. $\mathsf{ppd}_1 = (1-p)p^4$, $\mathsf{ppd}_2 = (1-p)p^3$, $\mathsf{ppd}_3 = (1-p)p^2 + p^5 + (1-p)^3p^2$, $\mathsf{ppd}_4 = 1$ and $\mathsf{ppd}_5 = p + (1-p)^2p$, and the expected ppd when taking into consideration all segments except s_4 is maximized for $p = 0.418$. The logic, as explained previously, is illustrated in Figure 5.3. The figure demonstrates that the robot can profit from turning around at s_4, since it can reach four segments, where if it travels deterministically through all segments, i.e., go straight to s_5, it will reach only three segments.

This strengthens our need to find the actual ppd values in all segments, since after obtaining these functions, it is possible to calculate the p value that maximizes the expected ppd throughout the line. This can be done using a procedure similar to ComputeProbPPD, while replacing the call to FindP to a function that calculates the p value that maximizes the expected ppd.

5.3 Handling an adversary with some knowledge in Perimeter Patrol

In the previous sections we have shown sound theoretical results for the optimality of patrol algorithms for an adversary that lies in one of the extremities of the knowledge scale (extremely strong or extremely weak). However, it is rarely the case that an adversary will not bear any *any knowledge* or will bear *full knowledge* on the patrolling robots. In most cases, the knowledge the adversary obtains on the robots lies somewhere on the knowledge continuum. In the following section, we provide a discussion on this case. First, in Section 5.3.1 we describe two heuristic algorithms to deal with such an adversary. In Section 5.4 we analyze the "some knowledge" adversary from the point of view of its uncertainty of its choice of action, and show theoretical optimality results for this case.

5.3.1 A heuristic approach - algorithm Combine

We have shown that the algorithm maximizing the expected ppd is the deterministic algorithm (Theorem 14). However, the deterministic algorithm creates high deviations between the ppd values in the segments: in t segments the ppd equals 1, and in the other $d - t + 1$ segments the value is 0. Figure 5.4 illustrates the tradeoff between using the deterministic algorithm ($p = 1$) and the MaxiMin algorithm ($p < 1$) for $d = 12, t = 9$. On the one hand, by maximizing the *expected* ppd, it creates a great deviation between the ppd values - either 1 or 0. On the other hand, by maximizing the *minimal* ppd, the ppd values decrease in many segments from 1 to lower values (compared to the deterministic algorithm).

In addition, the deterministic algorithm is easy to detect, therefore if the adversary has even a little time to study the system, it might deduce the

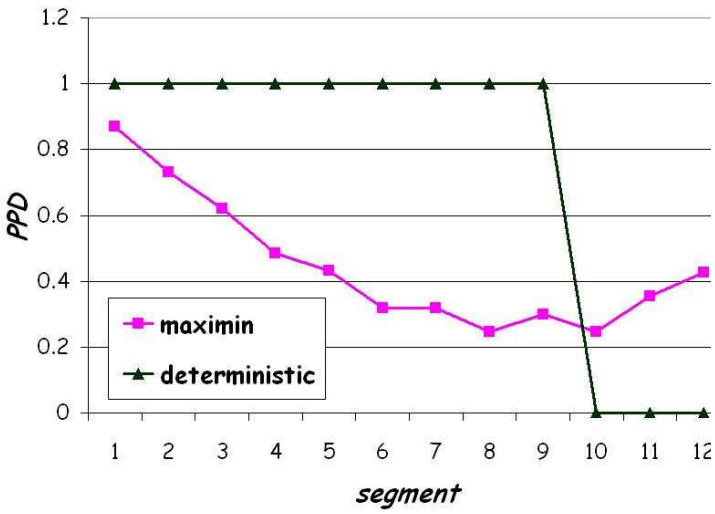


Figure 5.4: An illustration of the tradeoff between the probability of penetration detection in all segments ($d = 12, t = 9$) when preparing for a full knowledge vs. a zero knowledge adversary (p returned by algorithm MaxiMin and the deterministic algorithm, respectively)

type of algorithm being used and choose to penetrate through a segment with $\mathsf{ppd} = 0$, resulting in a successful penetration.

Motivated by the two reasons described above, we decided to adopt both a more "risk averse" approach and a method that will minimize the deviation between the ppd values throughout the perimeter. Therefore we present the $\mathsf{Combine}(d, t, w)$ algorithm that maximizes the expected ppd for the given d and t while minimizing the deviation between the ppd along the segments. This combination is done using the weight $w, 0 \leq w \leq 1$ for maximization of the expected ppd and the weight $1 - w$ for minimization of the deviation. A full description of Algorithm Combine is given in Algorithm 9. Note that this algorithm uses a procedure of Algorithm MaxiMin for finding the ppd in each segment. This procedure is dynamic-programming inspired, separates each state into two based on directionality (clockwise and counterclockwise) and by assigning values in a matrix it determines the ppd in a segment. An absorbing state is used in order to represent the fact that the ppd is determined only by the *first* visit to a segment. Denote the standard deviation between the ppd values of a vector of functions $F = \{f_1, \ldots, f_{d-1}\}$ by $\text{stdev}(F)$. Denote each segment $i, 1 \leq i \leq d-1$ by s_i.

Algorithm 9 $\mathsf{Combine}(d, t, w)$

Calculate F as follows:
for each $s_{init} = s_i \in \{s_1, \ldots, s_{d-1}\}$ **do**
  Create the matrix M of size $(2d+2) \times (t+1)$, initialized with 1 in $M_0(s_{init})$ and 0s otherwise, using the following rules.
  for each entry $M_t(s_i^{cw})$ **do**
    set value to $p \cdot M_{t-1}(s_{i+1}^{cw}) + q \cdot M_{t-1}(s_i^{cc})$
  for each entry $M_t(s_i^{cc})$ **do**
    Set value to $p \cdot M_{t-1}(s_{i-1}^{cc}) + q \cdot M_{t-1}(s_i^{cw})$
  for absorbing states **do**
    Set entry $M_t(s_{abs}) = M_{t-1}(s_{abs}) + p \cdot [M_{t-1}(s_1^{cw}) + M_{t-1}(s_d^{cc})]$
$F \leftarrow$ row t of M.
$Q_1 \leftarrow 1/d \sum i = 1^d F_i$
$Q_2 \leftarrow 1 - \text{stdev}(F)$
$Q \leftarrow wQ_1 + (1-w)Q_2$
Return $p = \max\{Q\}$

5.3.2 A heuristic approach - algorithm MidAvg

Another possible heuristic approach to dealing with an adversary with some knowledge, is to combine the optimal algorithm for the full knowledge adversary and the optimal algorithm for the zero-knowledge adversary. In other words, Algorithm MidAvg will return a p value that is a weighted combination of p returned from the MaxiMin algorithm, and $p = 1$. A full description of the algorithm is given below.

Algorithm 10 MidAvg(d, t, w)

1: $p_{full} \leftarrow$ FindP(d, t)
2: $p_{zero}\ gets 1$
3: $p_{weight} \leftarrow w \times p_{full} + (1 - w) \times p_{zero}$
4: Return p_{weight}

5.4 Theoretical results about adversarial uncertainty

In most cases, it is realistic to assume that the adversary's knowledge on the patrol algorithm lies somewhere along the knowledge continuum, between full and zero knowledge. Usually, the adversary does not gain enough information on the patrol algorithm in order to derive the exact algorithm (i.e., probability p) or the exact weakest spots of the algorithm. Therefore we theoretically explore two directions for handling partial knowledge of the adversary. In the first, the adversary might have some estimation of the probability p characterizing the patrol algorithm. In the second, the adversary might have some estimation of the weakest spot of the algorithm. In both cases, we wish to use the region of possible beliefs of the adversary in order to find an optimal patrol algorithm for the patrolling robots.

A common way of handling uncertainties of systems is to assume that when having no knowledge, a random choice, with uniform probability, is made. In this domain, this approach was proven to be useful in an empirical evaluation (see Chapter 6, where a patrol algorithm proven to be optimal for a random adversary performed substantially better than other algorithms for humans playing the role of an adversary that had no knowledge of the

patrolling robots. We will use a similar approach here, i.e., within the region of estimation of the adversary—either of the patrol algorithm p or of the weakest spots—the adversary will be assumed to choose its actions at random.

We first examine the approach in which the adversary estimates the probability p characterizing the patrol algorithm with some error. Unfortunately, we show that it is impossible to find an optimal patrol algorithm in this case.

We then discuss two alternative approaches, in which the uncertainty is reflected by the choice of the *penetration spot*. In this case, we do not necessarily assume that the adversary calculates the probability p, but tries to estimate the weakest spot using two estimation methods - physical proximity, or probability proximity to the minimal ppd.

5.4.1 Estimating p - negative result

In this section we discuss the case in which the adversary estimates the probability p characterizing the patrol algorithm.

The problem of estimating the probability p can be considered as observing a Bernoulli trial, where a success is an event of going straight with a probability of p, and a loss is turning around with a probability of $1-p$. We can use the Central Limit Theorem [26] that bounds the expected error from the real value of p after viewing it for t_v trials. The average of successes after viewing t_v trials will be within the boundaries of $[p-\delta, p+\delta]$ with a probability of p_{conf}, where δ is a function of t_v and depends on p_{conf}. Therefore, the adversary can estimate the real value of p inside some interval around p, and we will try to use this interval in order to optimize the patrol algorithm of the robots. Consider the following problem.

P-Interval **problem definition:** Let p be the probability characterizing the perimeter patrol algorithm of a team of robots. Assume the adversary estimates that the real value of p is inside the interval $[p-\delta, p+\delta]$. Therefore it chooses its believed p_b at random with a uniform probability within this interval. The algorithm needs to find the probability p characterizing the patrol of the robots such that it maximizes the expected ppd throughout the perimeter.

Unfortunately, we prove that this problem is unsolvable unless $\delta = 0$. We prove this by showing that the expected ppd function inside the interval $[0, 1]$ monotonically increases, i.e., as p grows the expected ppd increases, hence the optimal p does not converge unless $\delta = 0$ (since maximization of the

expected ppd is obtained in the right bound of the interval, which is also the midpoint of the interval only when $\delta = 0$).

Denote the number of times a robot switches directions during t time units by r, $r \geq 1$, and the addition to the ppd in segment s_j by some robot R_0 after switching its direction r times by $\mathsf{ppd}_j^0(r)$.

Lemma 18. *Consider a sequence of* $2d$ *segments with one robot* R_0 *in the mid segment at time* 0*. Then* $\sum_{j=1}^{2d} \mathsf{ppd}_j^0(r) < \sum_{j=1}^{2d} \mathsf{ppd}_j^0(r-1)$.

Proof. We divide the sequence of $2d$ segments into two: the sequence to the right of R_0 (with positive indexes) and to the left of R_0 (with negative indexes). For every j number of direction switches, let $\delta(j) = \sum_{i=-t+j+1}^{-1} \mathsf{ppd}_i^0$ (sum of R_0 contributions to the ppd functions to segments with negative indexes except for s_{-t+j}), $\delta'(j) = \mathsf{ppd}_{-t+j}^0$ (R_0's contribution to s_{-t+j}), and similar definitions for segments with positive indexes, i.e., $\alpha(j) = \sum_{i=1}^{t-j} \mathsf{ppd}_i^0$ and $\alpha'(j) = \mathsf{ppd}_{t-j}^0$.

The sum of ppd_i^0 for $r-1$ number of direction switches is $\delta(r-1) + \delta'(r-1) + \alpha(r-1) + \alpha'(r-1)$. For r switches, since the robots spend an extra time cycle to turn around, the two extreme segments with ppd > 0 are now unreachable, hence in this case $\delta'(r-1)$ and $\alpha'(r-1)$ no longer exist. Now, $\delta(r) + \delta'(r)$ is similar to changing the initial direction of the robot (by multiplying by $1-p$), and obtaining exactly $\alpha(r-1)$, hence $\delta(r) + \delta'(r) < (1-p)\alpha(r-1)$. Similarly, $\alpha(r) + \alpha'(r) < (1-p)\delta(r-1)$. Altogether, $\sum_{l=1}^{2d} \mathsf{ppd}_l^0(r) = \delta(r) + \delta'(r) + \alpha(r) + \alpha'(r) < (1-p)\alpha(r-1) + (1-p)\delta(r-1)$ and since $(1-p) < 1$ this is smaller than $\sum_{l=1}^{2d} \mathsf{ppd}_l^0(r-1)$. By the induction assumption, this is smaller than t. The proof follows directly for $\tau > 1$, as the number of segments that become unreachable increases from 1 to τ for each direction switch, while the probability of penetration detection in other segments remains the same. □

Let $E_{\mathsf{ppd}}(p)$ be the expected ppd for probability $p \in [0,1]$ (the probability that the robots will continue straight in each time unit during the patrol) $0 \leq p \leq 1$,

Lemma 19. *The expected* ppd*, as a function of* p*, is a monotonically increasing function in the range* $[0,1]$*, i.e., for all* $0 \leq p' < p \leq 1$*,* $E_{\mathsf{ppd}}(p') < E_{\mathsf{ppd}}(p)$

Proof. Denote the expected number of direction switches of robot R during t time units using probability p of going straight by $E_{sw}(p)$. Therefore $E_{sw}(p) = t(1-p)$ and $E_{sw}(p') = t(1-p')$, and since $p' < p$ it follows that $E_{sw}(p) < E_{sw}(p')$.

Now we must show that if a robot R is expected to switch its direction more times during t time units, then the expected ppd will be smaller. Formally, we want to show that for $0 \leq p' < p \leq 1$, $E_{sw}(p) < E_{sw}(p') \Rightarrow E_{\mathsf{ppd}}(p') < E_{\mathsf{ppd}}(p)$.

The expected ppd along the perimeter with r direction switches is $E^r_{\mathsf{ppd}} = 1/N \sum_{i=1}^{N} \sum_{j=1}^{k} \mathsf{ppd}^k_i(r)$. During $t < d$ time units the robot can influence the ppd along the perimeter at most $2d$ segments, hence the sum of each robot is not over all N segments, but only the neighboring d segments on each of its sides. Therefore, following Lemma 18, $E^r_{\mathsf{ppd}} < E^{r-1}_{\mathsf{ppd}}$. Thus if $E_{sw}(p) < E_{sw}(p')$ then $E_{\mathsf{ppd}}(p') < E_{\mathsf{ppd}}(p)$. □

Theorem 20. P-Interval *is unsolvable unless* $\delta = 0$.

Proof. Assume, for the purpose of contradiction that $\delta > 0$, yet $p*$ that maximizes the expected ppd throughout the perimeter exists. By the definition of P-Interval, the adversary deduces an interval around $p*$ whereby it chooses its believed p at random inside the interval $[p* - \delta, p* + \delta]$. By Lemma 19, the expected ppd function monotonically increases, therefore the maximal expected ppd inside this interval is obtained in $p* + \delta$. This contradicts the assumption that $p*$ maximizes the expected ppd, unless $\delta = 0$. □

5.4.2 Uncertainty in the choice of the penetration spot

In this section we explore the case in which the partial knowledge of the adversary on the patrol algorithm is translated into different possible options of the penetration spots. For several reasons, the adversary might not choose to penetrate through the *exact* weakest spot. In this section we present two deviations from the weakest spots, and consequently two possible corresponding optimal ways of choosing the patrol algorithm in such cases.

The adversary, after studying the robots' patrol for a period of time, could determine several reasonable segments in which the ppd values, as it believes, are small enough. In this case it could choose to penetrate through one of the v weakest spots at random, with some probability distribution (for example uniform). Hence the robots should choose p such that the expected

ppd along the v segments with minimal ppd is maximal. We refer to this approach as v-Min.

The second case is that the adversary might not choose to penetrate through the segment with the minimal ppd, but either through that segment, or through one of its neighboring segments at random. Hence the robots should choose p such that the minimal expected ppd along v neighboring segments is maximized. This approach is referred to as v-Neighbor.

Note the difference between the two cases - in v-Min we are looking for the value $0 \leq p \leq 1$ such that the weighted average of the v minimal ppd's is maximized, and in the v-Neighbor case we are looking for p such that the minimal weighted average of v neighboring segments is maximized.

In both cases, the two extremities of uncertainties—full knowledge adversary (no uncertainty) and zero knowledge adversary (complete uncertainty)—match the results presented in Sections 5.1 and 5.2, respectively. If $v = 1$, i.e., there is no uncertainty in the choice of the weakest spot, then the algorithms are required to precisely return the value p such that the minimal ppd is maximized, similar to the MaxiMin algorithm presented in Section 5.1. On the other hand, if $v = d$ and the probability distribution is uniform, then the algorithms will return the value p that maximizes the expected ppd throughout the perimeter (=average ppd). As proven in Section 5.2, the optimal algorithm in this case is $p = 1$, i.e., the deterministic algorithm.

The algorithms used to find an optimal patrol use the ppd_i function for each segment s_i. The ppd_i is a function of p, and it is calculated in polynomial time using a dynamic-programming algorithm described in Chapter 4.

Optimality of the patrol algorithm using the v-Min approach

In this sectin we present the ComputeMinV algorithm, which finds the optimal patrol algorithm, corresponding to the probability p of going straight at each time step in the v-Min scenario. Specifically, the ComputeMinV algorithm computes the value p such that the minimal v ppd's are maximized, given a probability distribution $V = \{v_i, v_2, \ldots, v_v\}$, where v_i is the probability that the adversary will choose to penetrate through the i'th weakest spot, $\sum_{i=1}^{v} v_i = 1$. This distribution can be used to further manipulate the impact of the extent of knowledge of the adversary on its choice of penetration spot. For example, after the adversary obtains more knowledge v_1 may increase to more than the uniform distribution $(1/v)$.

The algorithm ComputeMinV operates as follows. First, it identifies all

intersection points between every pair of ppd_i, ppd_j functions ($1 \leq i, j \leq d$, $i \neq j$). Then it divides the range $[0, 1]$ into sections according to all the intersection points. For each section $[p_a, p_b]$, the algorithm identifies the minimal v curves between $[p_a, p_b]$, and finds their average curve, f_{avg}. Since the adversary chooses to penetrate through one of the v segments with the lowest ppd at random with a given distribution V, the *weighted average* (given weight v_i to the i'th minimal curve) of the v curves represent the *expected* ppd in that section. Last, ComputeMinV calculates the maximal value of $f_{avg}(a, b)$ in the section $[p_a, p_b]$, and reports the point p_{opt} that is maximal among all minimal points of the average functions. An illustration of this algorithm is shown in Figure 5.5.

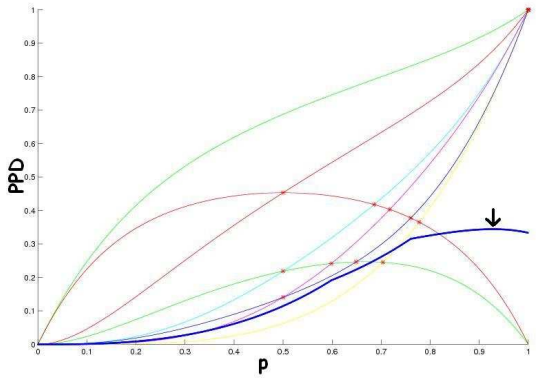


Figure 5.5: An illustration of the ComputeMinV algorithm for $d = 8, t = 6, v = 3$. The small stars mark the intersection points, and the bold curve is the average of the 3 minimal curves at each section. The arrow marks the maximal point computed by ComputeMinV.

The time complexity analysis of the ComputeMinV algorithm is as follows. There are at most d intersection points between every two pairs of curves (all curves are polynomials of p of order of at most d, hence there are at most d real roots to the subtraction of both polynomials), therefore altogether there are at most d^3 intersection points. The algorithm sorts all intersection points, in a time complexity of $\mathcal{O}(d^3 \log d^3)$ (using any standard sorting algorithm). In each section the algorithm invests dv in finding the minimal v curves, hence (assuming v is a constant), the total complexity of this part is at most d^4. Altogether the total time complexity is $\mathcal{O}(d^4 + d^3 \log d^3)$, compared to a time complexity of d^3 of the original MaxiMin algorithm for full knowledge

Algorithm 11 ComputeMinV$(v, V, \{\mathsf{ppd}_1, \ldots, \mathsf{ppd}_d\})$

1: Set $BufP \leftarrow \{0, 1\}$ {initialize list of all intersection points}
2: **for** every pair $\mathsf{ppd}_i, \mathsf{ppd}_j$, $1 \leq i, j \leq d$, $i \neq j$ **do**
3: $Intersect_{i,j} \leftarrow$ intersection points between ppd_i and ppd_j.
4: $BufP \leftarrow BufP \bigcup Intersect_{i,j}$
5: Sort $BufP$ in ascending order
6: $Res_f, Res_p \leftarrow 0$ {initialize maximin value and its p }
7: **for** $j \leftarrow 1$ to $|BufP|$ **do**
8: Find v functions $f_{j_1}, \ldots, f_{j_v}$ such that $f_{j_i}(p') < f_n(p')$ $\forall p' \in [BufP(j), BufP(j+1)], 1 \leq i \leq v$, $f_n \neq f_{j_i}$
9: $f_{avg} \leftarrow \sum_{i=1}^{v} v_i \times f_{j_i}$
10: $m \leftarrow f_{avg}(p*)$ such that $\forall p \in [BufP(j), BufP(j+1)]$, $f_{avg}(p*) \geq f_{avg}(p)$
11: **if** $m > Res_f$ **then**
12: $Res_f \leftarrow m$; $Res_p \leftarrow p*$
13: Return Res_p

adversary.

Optimality of the patrol algorithm using the v-Neighbor approach

As stated previously, the adversary might attempt to penetrate not only through the weakest segment, but through one of its neighboring segments. Therefore this can be used in order to find a patrol algorithm (p value) more suitable for the situation. Algorithm ComputeNeighborV computes the weighted average of v neighboring segments according to a distribution $V = \{v_1, \ldots v_v\}$, then finds the maximin point of the new curves. Note that if the robot currently resides inside the v-neighborhood of a segment s_i (i.e., $v - i < 0$ or $v + i > d$), its current location is excluded, i.e., we average less segments for that case. The probability distribution can be used to express the fact that the adversary tends, for example, to try to penetrate through the segments further away from the robot in its current position. Figure 5.6 illustrates the algorithm for $d = 8$, $t = 6$ and $v = 3$.

The time complexity of Algorithm ComputeNeighborV is $\mathcal{O}(d^3)$, since finding the weighted average of v neighboring ppd functions costs vd, and finding the MaxiMin of the d functions is $\mathcal{O}(d^3)$ (see Section 5.1). Consequently the total time complexity of the ComputeNeighborV algorithm is $\mathcal{O}(d^3 + d) = \mathcal{O}(d^3)$

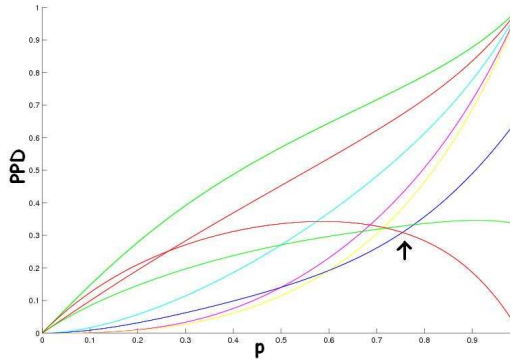


Figure 5.6: An illustration of the ComputeNeighborV algorithm for $d = 8, t = 6, v = 3$. The curves are *not* the original ppd_i functions, but the average of the $v-$neighborhood of each segment. The arrow points to the maximin point of the new curves.

Algorithm 12 ComputeNeighborV$(v, V, \{\mathsf{ppd}_1, \ldots, \mathsf{ppd}_d\})$

Set $FuncSet \leftarrow \emptyset$
for $i \leftarrow 1$ to d **do**
  $i_e = \min(d, i + v)$
  $FuncSet \leftarrow \sum_{j=i}^{i_e} v_{j-i+1} \times \mathsf{ppd}_j \bigcup FuncSet$
$p_{opt} \leftarrow$ MaxiMin$(FuncSet, d)$
Return p_{opt}

(similar to the complexity of MaxiMin).

Comparing v-Min and v-Neighbor

The two approaches of v-Min and v-Neighbor towards bounding the uncertainty of the adversary in its choice of penetration spot seem to be coherently different. Consider for example the case in which $d = 8$, $t = 6$ and $v = 3$ (Figures 5.5 and 5.6). The optimal p returned by v-Neighbor is $p = 0.7359$, and the optimal p according to v-Min is $p = 0.9273$. The result returned by the MaxiMin algorithm (used in cases of a full knowledge adversary, i.e., $v = 1$) is $p = 0.7037$.

However, in some cases they coincide, as proven in the following Theorem. Below we discuss such cases, in order to provide a better understanding of this cohesion. The following Theorem holds for every probability distribution V, but for simplicity reasons we prove it for $v_i = 1/v$, i.e., using a uniform distribution inside the bounds of uncertainty.

Theorem 21. *The optimal choice of p according to* v-Neighbor *coincides with the optimal p according to* v-Min *if $t = \lfloor d/2 \rfloor + 1$.*

Proof. The optimal p in the $v-$neighborhood and in the $v-$ minimal, is the one which maximizes the minimal ppd of the average of the $v-$ neighborhood and $v-$ minimal ppd_i functions, correspondingly. Therefore it is suffice to show that along this optimal point p_o, the $v-$ minimal ppd_i functions are also all neighbors. Formally, we need to show that $\mathsf{ppd}_{i_1}, \ldots, \mathsf{ppd}_{i_v}$ are minimal, where $i_l = j + l$ for some index $1 \leq j \leq d - v$.

Consider the section of d segments between two consecutive robots R_a and R_b. First, assume d is odd. In this case, ppd_i for $1 \leq i \leq t$ is influenced only by R_a, and ppd_i for $t + 1 \leq i \leq d$ is influenced only by R_b. Moreover, every ppd_i function for $1 \leq i \leq t$ equals 0 if $p = 0$, and equals 1 if $p = 1$. On the other hand, every ppd_i function for $t + 1 \leq i \leq d$ equals 0 in both cases where $p = 0$ and $p = 1$.

Note that if a robot is headed clockwise, then any ppd function of a segment s_i at a distance of i to its right is larger than a ppd function of a segment which is at the same distance, but to the left. For example, $\mathsf{ppd}_1 > \mathsf{ppd}_d$, $\mathsf{ppd}_2 > \mathsf{ppd}_{d-1}$ and so on. The reason lies in the fact that the probability of reaching a segment of a distance if i in the opposite direction is equivalent to the probability of reaching a segment at a distance of i in

the same direction, but multiplied by $(1-p)$. Since we assume that $p \leq 1$, this is always true.

Combining all known facts together, from Lemma 1 we see that $\mathsf{ppd}_{t+1} \leq \mathsf{ppd}_{t+2} \leq \ldots \leq \mathsf{ppd}_{d-1} \leq \mathsf{ppd}_d$ and $\mathsf{ppd}_t \leq \mathsf{ppd}_{t-1} \ldots \leq \mathsf{ppd}_1$. Also, as shown herein, $\mathsf{ppd}_1 \geq \mathsf{ppd}_d$, $\mathsf{ppd}_2 \geq \mathsf{ppd}_{d-1}, \ldots, \mathsf{ppd}_{t-1} \geq \mathsf{ppd}_{t+1}$. It follows that, obviously, the minimal function is ppd_{t+1}, the function above it is ppd_t and ppd_{t-1}, followed by ppd_{t+2} and ppd_{t-2} and so on (see the example in Figure 5.7). Therefore, inevitably, when considering $v-$ minimal segments, for all v, we remain in the $v-$ neighborhood of ppd_{t+1}.

If d is even, then the only difference is that function ppd_t receives components from both R_a and R_b, and thus it is not straightforward that it is smaller than ppd_{t-1}. Calculating the exact value of ppd_t shows us that $\mathsf{ppd}_t = p^t + (1-p)p^{t-1} = p^{t-1}$. On the other hand, $\mathsf{ppd}_{t-1} = p^{t-1}$, i.e., $\mathsf{ppd}_{t-1} = \mathsf{ppd}_t$, and the rest of the proof follows directly as in the case of an odd d. □

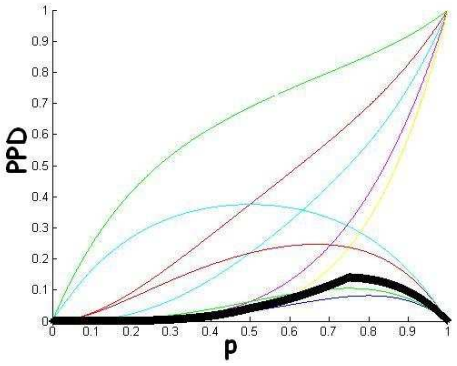


Figure 5.7: An illustration of proof of Theorem 21, in which the $v-$neighborhood and $v-$ minimal coincide ($d = 9$, $t = 5$ and $v = 3$). The bold line represents the average of $v-$ minimal / $v-$ neighboring segments.

Chapter 6

Empirical Evaluation

In this chapter, we describe an empirical evaluation of the performance of the different patrol algorithms working against adversaries that obtained different amounts of knowledge on the patrolling robots. The evaluation is made using the Penetration Detection game (PenDet-Game) we created. In this game, the adversary's role is played by human subjects, working against simulated robots. In Section 6.1 we describe the PenDet-Game. In section 6.2 we present the first setting of the game, in which we compared the performance of the deterministic algorithm, the MaxiMin and Combine against three different amounts of exposed information. In Section 6.3 we concentrate on evaluating adversaries with *some* knowledge, using the MaxiMin, MidAvg and the family of v-Neighbor and v-Min algorithms.

6.1 The PenDet-Game

In the PenDet-Game, a human player plays the role of the adversary, working against a team of simulated patrolling robots. Therefore the player is required to pick a segment through which he thinks he can penetrate without being detected.

Note that our choice of performing experiments in this simulated environment, rather than actual robots is not trivial. The reason for preferring to conduct such experimental research, is that managing to evade patrolling robots using current lab-robots is simple—the adversary can simply jump over them. Moreover, extensive experiments were required. to evaluate performance of the patrol algorithms. This is again impossible to create with real

robots. Note that there are empirical results from running experiments with real robots in systems with adversarial teams, e.g. the Robocup game [52], however they were conducted between two teams of robots, not humans vs. robots.

The game consists of four robots patrolling around a treasure pot. On the game screen, the player can see the circle representing the perimeter and the patrolling robots (Figure 6.1). The distance between the robots and the time it takes to penetrate change from one subgame to the other. These values are presented explicitly to the player throughout the subgame. For simplicity reasons, we designed the game with $\tau = 1$, i.e., each time the robots switched directions they stayed in the same segment during that time cycle.

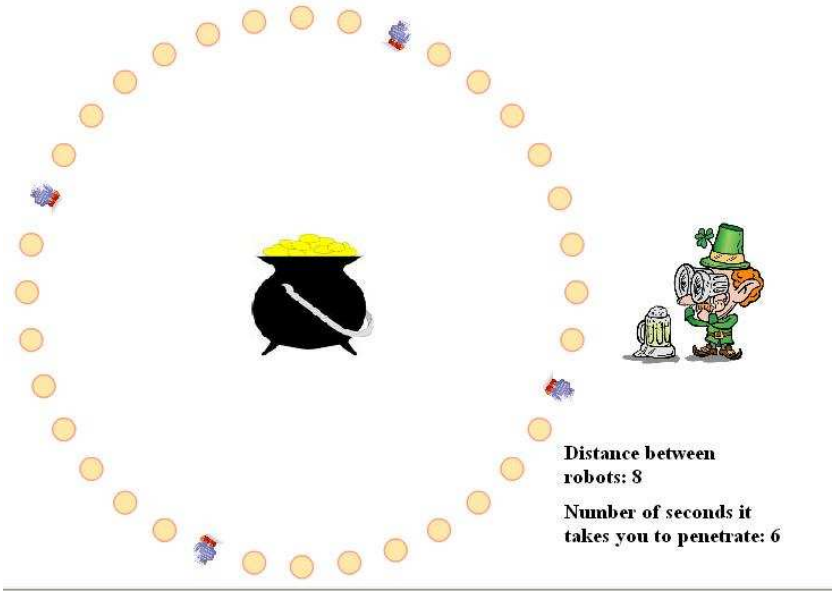


Figure 6.1: The PenDet-Game screen.

6.2 Experiment - Phase 1

The game consisted of three stages, where in each stage the player had more time to study the system, i.e., more information concerning the patrolling robots was revealed gradually to the player. We describe the game in detail in the following paragraphs.

In order to simulate different knowledge states, the game had three stages.

1. In the first stage, the player was shown a static picture with the current location of the robots. The direction of the robots (where they were facing) could be easily deduced from the picture of the robots. The player was requested to choose the segment through which he believed he would be able to penetrate without being detected. This stage consisted of three sub-games. In each sub-game the distance between the robots and/or the penetration time t were different.

2. In the second stage, the player was shown five seconds of the patrol. After these five seconds passed the player was requested to click on the segment through which he thought he had the best chances of penetrating without being detected. No feedback was given to the player regarding whether he succeeded or failed in his attempt to penetrate. This stage consisted of six sub-games. In each sub-game the patrol scheme of the robots, the distance between the robots and/or the penetration time t were different.

3. The third stage of the game was a three minute game, in which the player could try to penetrate as many times as he wanted (as long as time permitted) simply by clicking on the section through which he decided to penetrate. The player could see whether he succeeded or failed in his attempt. This stage also consisted of six sub-games. In each sub-game the patrol scheme of the robots, the distance between the robots and/or the penetration time t were different.

Each game (and all its subgames) was played *once* by each player, so that the primacy of the choices taken by the players in the first stages was maintained.

The PenDet-Game was played by 68 human subjects (29 Female/39 Male). All subjects were senior undergraduate students in computer science. The game was set online, and the students were required to play it as part of their course requirements.

The information regarding the d, t, p values tested, is given in Table 6.1.

Three different patrol algorithms were executed in three stages of the game. The patrol algorithm was determined by the probability p characterizing the robots' movement. The first algorithm corresponded to the zero

knowledge adversary, therefore following Theorem 14 this was the deterministic patrol algorithm ($p = 1$). We denote this algorithm by Det. Note that the player did not completely have *no knowledge* of the patrol scheme, since he knew the distance between the robots and the direction it they were facing. However, this information was minimal and did not reveal anything concerning the patrol algorithm. The second algorithm was the Combine algorithm. The third algorithm corresponded to the full knowledge adversary. Note that the player did not have *full* knowledge, but did receive a long period of time to study the system, which brought the player close to a full knowledge adversary. In this case, the p values represented the probability yielding the maximal minimal ppd along the perimeter. The values of p were calculated using the MaxiMin algorithm described in Section 5.1.

Det was executed in the first and second stages of the game, where MaxiMin and Combine were executed in all three stages of the game. The reason for omitting Det from the third stage is threefold. First, the combined algorithm reaches 1 as t gets close to d, and is exactly 1 (deterministic) if $t = d - 1$. Therefore checking two cases of deterministic algorithms would give a clear picture of this behavior. Second, we assumed that in this case the learning curve of the players would be steep, i.e., they would understand that this is a deterministic algorithm and succeed in nearly all attempts to penetrate. Last, we thought that this algorithm might bore the players and eliminate their motivation to play thoughtfully. This was verified by feedback we received from subjects in the early developmental stages of the game.

In order to evaluate the performance of the three algorithms, we executed the algorithms on the input retrieved from the choices of the players' penetration spots. In the first stage, each player provided 3 input lines, each compatible to one pair of d and t. For each such input line, we calculated

Table 6.1: The d, t, p values tested in the experiment.

d	t	p Det	p Combine	p MaxiMin
16	9	1	0.93	0.87
8	5	1	0.92	0.75
8	6	1	0.96	0.7
12	9	1	0.97	0.77
12	11	1	1	0.82
16	15	1	1	0.85

the respective probability of penetration detection (determined by the chosen penetration spot)—according to the three different algorithms, i.e., each input line produced data for each one of the three algorithms. In the second stage, each input from each player corresponded to a triplet, d, t, p. By the choice of the penetration spot, we determined the probability of penetration detection of the algorithm (characterized by p). In the third stage, we calculated the probability of penetration detection for each penetration attempt according to the relative position between the robots at the time the player made his decision. We then took the average of the probability of penetration detection over all penetration attempts of the player, thus each player contributed one input line to the analysis for each subgame in this stage.

6.2.1 Experimental results

In this section, we describe the results of the experiments with the PenDet-Game. First we describe the bottom line summary of the results, then we discuss the results of each stage of the game in detail.

Figure 6.2 describes the summary of the results obtained from all three stages of the game. It presents the maximal, minimal and average penetration detection ratio obtained by each of the algorithms we tested each stage of the game. Comparing these values, it is clear that in the first stage the deterministic algorithm is the best: its average is considerably higher than the average of the other algorithms, and the minimal penetration detection is also considerably higher. Note that the maximal value of penetration detection is equal to the maximal value of the Combine algorithm, since this value results from the Combine algorithm when it offers deterministic behavior. In the second stage, the average value of penetration detection obtained by all three algorithms is relatively similar, yet the Combine algorithm is considerably better than the deterministic algorithm in its minimal value, and substantially better than the MaxiMin algorithm in its maximal value. In stage 3, MaxiMin significantly outperforms the Combine algorithm in both average and maximal penetration detection. The minimal penetration detection is similar, due to the fact that when t is small relative to d, MaxiMin cannot guarantee high ppd values, thus they are similar to the ppd guaranteed by the Combine algorithm.

The game results are mainly evaluated in terms of actual percentage of penetration detection from all three algorithms, which corresponds to the robots' performance in different scenarios. In some cases, we found that the

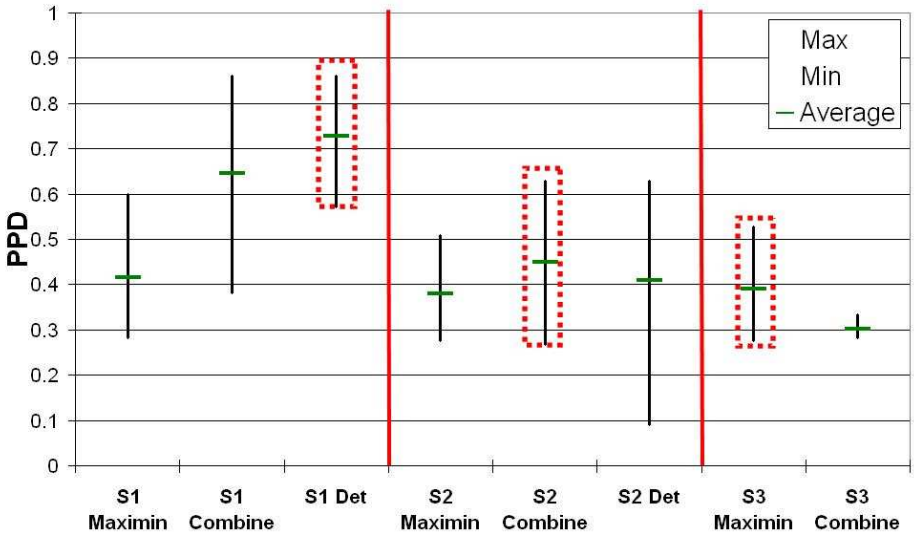


Figure 6.2: A summary of the results, divided into three stages: no information ($S1$), short-term revelation of information ($S2$), and long term revelation of information ($S3$) for the three patrol algorithms. Each line represents the maximal, minimal and average penetration detection. The best performing algorithm in each stage is depicted by a surrounding dotted rectangle.

choice of the player of the section through which he decided to penetrate yielded interesting results. This corresponds to the decisions taken by the adversary after attaining different levels of information.
Stage 1*:*
In the first stage, nearly no information was given to the player. Therefore the players could have chosen the penetration spots at random. In Figure 6.3 we see that, however, in most cases the players chose to penetrate through one of two segments in the middle. This is not surprising, considering that people are drawn to central positions when instructed to choose between positions that have no apparent special characteristics [54]. In addition, the direction of the robot is visible to the player, hence he might take that into consideration. This is apparent in the case where $d = 12$ and $t = 11$, in which 28% of the players chose to penetrate through the last segment.

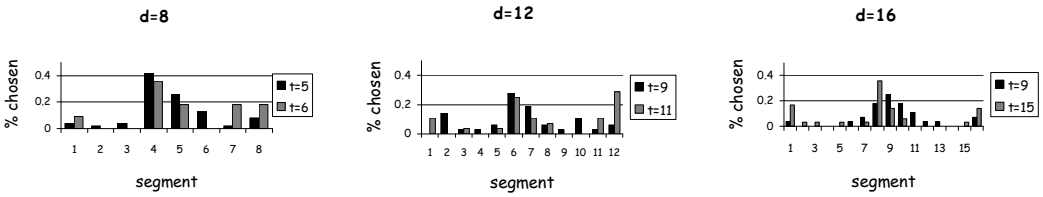


Figure 6.3: Choices of penetration positions in stage 1 for different values of d: $d = 8, 12, 16$. The x axes represents the segment, and the y axes the percentage of subjects that chose to penetrate through that segment.

As clearly depicted in Figure 6.4, the algorithm that managed to detect the highest percentage of penetrations is the deterministic algorithm. The data was examined statistically using a special macro designed by Brunner et al. [14] for running the Friedman test with repeated measures (the data we have does not have normal distribution, thus we needed a nonparametric test). The advantage of using the deterministic algorithm compared to MaxiMin and Combine algorithm was found to be statistically significant, with $p - value = 0.04$ for Combine vs. the deterministic algorithm, and $p - value = 0.003$ for MaxiMin vs. the deterministic algorithm. Therefore the deterministic algorithm is indeed more suitable for detecting penetrations in case the adversary has nearly no knowledge of the patrol scheme. Moreover, even if the adversary has some knowledge - in our case the distance between the robots and the direction they are currently heading - this algorithm still performs nicely. However, the expected values of ppd ("theoretical expected

ppd" in Figure 6.4) are higher than what was obtained in the actual game. The reason for this outcome is that the algorithm maximizes the expected ppd if the adversary chooses its penetration spot at random. However, as we have seen previously, this is not necessarily the case (in other words: it expects a less sophisticated adversary). On the other hand, the MaxiMin algorithm expects to be teamed up against a much more sophisticated adversary, therefore the actual penetration detection percentage is higher than the theoretical values. Note that the Combine algorithm coincides with the deterministic algorithm for some scenarios ($t/d = 11/12, 15/16$), therefore the penetration detection percentages of both are identical in those cases.

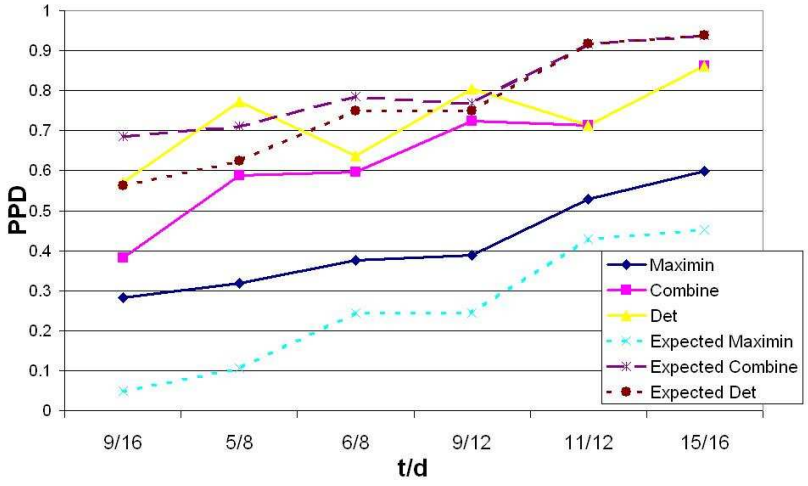


Figure 6.4: Performance of the three different algorithms in stage 1 (adversary with nearly zero knowledge).

Stage 2:
When only a small amount of information was revealed to the player concerning the patrol scheme (5 seconds), then the Combine algorithm performed better compared to the other algorithms based on the analysis shown in Figure 6.2. As presented in that figure, the average of ppd obtained the Combine algorithm is slightly higher than the average ppd obtained by MaxiMin and Det. Moreover, we note that the minimal ppd of Combine is higher than the minimal ppd of Det, and the maximal ppd of Combine is higher than the maximal ppd of MaxiMin. However, as clearly demonstrated in the detailed results presented in Figure 6.5, all the results are not statistically significant.

We can therefore only conclude that the deterministic algorithm is no longer optimal (as shown in the first stage, with zero-knowledge adversaries), and that Combine performs better compared to it. The reason for this lies in the fact that although only a small amount of information was exposed to the player, in most cases it was enough to determine that if the robots might turn around, then it is not beneficial to penetrate through the last $d - t$ segments. However, after 5 seconds of observation a player cannot infer the patrol algorithm and identify weakest spots or differ between algorithms (in this case the Combine and MaxiMin algorithms).

Note that the theoretical values of ppd in both Combine and Det are considerably higher than the actual penetration detection ratio. This is clear, since the robots expect an adversary with no knowledge about the system, yet confront an adversary that has gained some information. On the other hand, the theoretical values of the MaxiMin still pose as a lower bound to the performance of the robots.

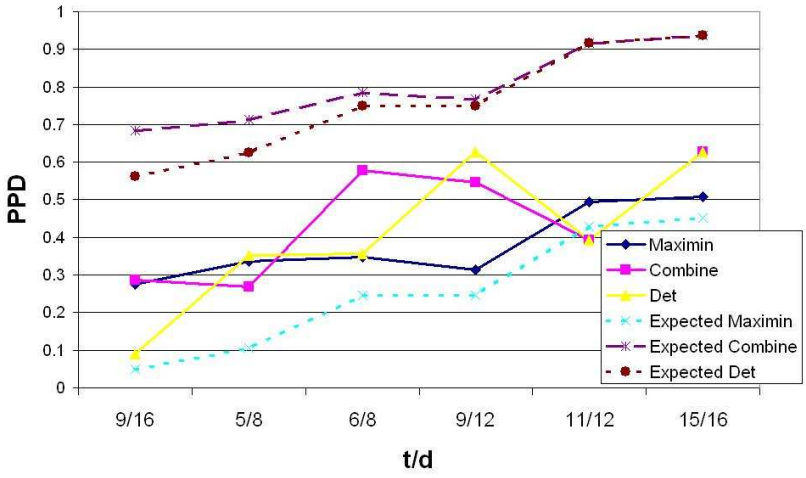


Figure 6.5: Performance of the three different algorithms in stage 2 (adversary with little knowledge)

Stage 3:
We present the results from stage 3 in two ways: the overall performance and the performance after omitting the first 30 seconds of the game. We consider the first 30 seconds to be a learning period, mainly for the Combine algorithm, when it produces a deterministic schedule.

First, the overall probability of penetration detection using the MaxiMin algorithm is statistically significantly better than the Combine algorithm even for the general performance over the entire 3 minutes (the difference is stronger after omitting the first 30 seconds). We used the Friedman test since the data does not have normal distribution, with $p - value < 0.0001$.

The following results are obtained when comparing the performance of the robots after omitting the first 30 seconds. When using the Combine algorithm, then when the algorithm is deterministic the penetration detection decreases from 30% or more to approximately 20%. Therefore even when the adversary observed the patrol for only 30 seconds, it managed to substantially increase its chances of successful penetration. In fact, the penetration detection ratio when using the MaxiMin algorithm is significantly better compared to the case in which the robots execute the Combine algorithm. This fact is even more interesting since we assumed that when the penetration time t would be higher, the robots would be more likely to detect the penetration (as clearly seen for the MaxiMin for all d, t pairs and for Combine in all non-deterministic cases without removing the learning phase). However, since the deterministic patrol scheme is simple and easily detected, when used even with high values of t, the adversary takes advantage of this and manages to penetrate with a higher probability. This fact again strengthens the motivation to concentrate on the case in which the adversary has some knowledge, and find suitable patrol schemes for this case. Note that the theoretical MaxiMin ppd still guarantees a lower bound to all non deterministic behaviors tested in this work. On the other hand, the MaxiMin algorithm performed generally the same along the entire 3 minutes and after omitting the first 30 seconds. The reason is clear: The players did not manage to deduce the patrol algorithm after 30 seconds, thus they could not improve their strategy against the patrolling robots.

6.3 Experiment - Phase 2

In the second phase of the experiment we concentrated on an adversary having *some* knowledge. Therefore the game consisted of several subgames, where in each subgame the player had time to study the system, i.e., information concerning the patrolling robots was revealed to the player, after which he had to choose his penetration spot. We describe the game in detail in the following paragraphs.

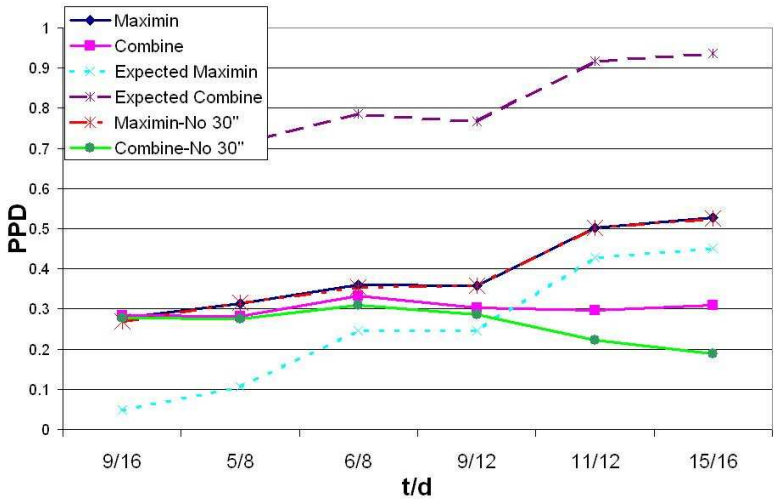


Figure 6.6: Performance of the two different algorithms in stage 3 (adversary with full knowledge) along the entire 3 minutes, and when omitting the first 30 seconds.

Each game consisted of several subgames. In each subgame, the player was given 60 seconds (equal to 60 time steps) of an *observation phase*. After the 60 seconds passed, the player was requested to click on the segment through which he thought he had the best chances of penetrating without being detected. No feedback was given to the player regarding whether he succeeded or failed in his attempt to penetrate.

The game was executed in two different versions, where each human subject played either the first version of the game, or the second version of the game. Altogether, 149 human subjects participated in this phase of the game. All subjects were either computer science undergraduate students, graduate students or alumni. The game was set online, and the student players were required to play it as part of their course requirements.

In the first version of the game, each game consisted of 6 subgames. In all subgames the patrol algorithm was unknown to the player, and the only information presented to him was the distance between the robots and the time it takes him to penetrate. This version of PenDet-Game was played by 71 human subjects.

The second version of the game was divided into two stages, each consisting of 6 subgames. The first stage included six subgames, similar to the

version described previously. After the first stage ended and before beginning the second stage, the player was informed of the general patrol algorithm. Specifically, it was told that the patrol algorithm instructs the robots to continue straight with a probability of p or turn around with a probability of $1-p$, and that this value of p remains the same throughout the subgame (yet changes between subgames). During the subgames themselves, the p value characterizing the patrol algorithm was presented to the player, during again 60 second observation period. This version of PenDet-Game was played by 78 human subjects (not the same subjects that played the previous version).

In each subgame the patrol scheme of the robots, the distance between the robots and/or the penetration time t were different. Each game (and all its subgames) was played *once* by each player. The players were instructed not to infer the patrol algorithm from one subgame to the other, as the patrol algorithms changed from one subgame to the other.

The information regarding the d, t, p sets tested, is given in the Table 6.1.

Table 6.2: The d, t, p values tested in the experiment.

Set number	d	t	p	Algorithm
1	8	6	0.7037	MaxiMin
2	8	6	0.7775	v-Min, $v = 2$
3	8	6	0.9273	v-Min, $v = 3$
4	8	6	0.85185	MidAvg
5	8	6	0.7604	v-Neighbor, $v = 2$
6	8	6	0.9095	v-Neighbor, $v = 3$
7	16	9	0.875	MaxiMin
8	16	9	0.8522	v-Min/v-Neighbor, $v = 3$
9	16	9	0.8329	v-Min/v-Neighbor, $v = 5$
10	16	9	0.8694	v-Min/v-Neighbor, $v = 7$
11	16	9	0.9561	v-Min/v-Neighbor, $v = 9$
12	16	9	0.9375	MidAvg

The experiment comprised 12 different sets. We considered two pairs of d, t values: $d = 8, t = 6$ and $d = 16, t = 9$, and different patrol algorithms for each such pair. The patrol algorithms used were determined as follows.

The patrol algorithm was determined by the probability p characterizing the robots' movement. Sets 1 and 7 corresponded to the full knowledge

adversary. In this case, the p values represented the probability yielding the maximal minimal ppd along the perimeter. The values of p were calculated using the MaxiMin algorithm described in Section 5.1. In addition, we checked the heuristic algorithm MidAvg in both d, t pairs (sets 7 and 12).

In sets 2 and 3 we use the v-Min algorithm to determine the p value, for $v = 2$ and 3 (respectively). Note that for $v > 3$ (in both cases) the deterministic algorithm was optimal, hence we did not examine it here. Similarly, in sets 5 and 6 used the v-Neighbor algorithm to determine the p value, for $v = 2$ and 3 (respectively).

In cases in which $d = 16$ and $t = 9$, then following Theorem 21, the v-Neighbor and v-Min algorithms output coincide. Therefore we sampled several v values: $v = 3, 5, 7, 9$ (sets $8, 9, 10, 11$, respectively).

In order to evaluate the performance of the algorithms, we referred to the penetration spots chosen by the players. We then calculated the *expected* probability of penetration detection according to the p values chosen by the algorithms.

Preliminary experiments also included observation periods of 5 and 30 seconds. However, results from these experiments—both from the choices made by the players and by the feedback given by them after playing the game—showed that these observation periods were not long enough to enable the players to understand of the system. Therefore the choices made by the subjects were arbitrary. As a result we focused on the experiments with an observation period of 60 seconds. Note that the first round of experiments, as reported in the Section 6.2.1, also strength this result. Namely, in the second stage of the game, in which the player had a 5 second observation period, no significance was strongly seen in respect to the optimality of patrol algorithms. We deduced, both by the choices made by the players and by their feedback, that the only information gained compared to the zero-knowledge stage (stage 1), was the possibility that the robots would turn around. In this experiment we required them to learn (in some manner) the differences between various nondeterministic algorithms, which apparently was impossible given the short observation periods of 5 and 30 seconds.

6.3.1 Experimental results and discussion

Analysis for unknown p:
Figure 6.7 describes the *expected* probability of penetration detection given

the players' choice of penetration locations for $d = 8, t = 6$ and $d = 16, t = 9$ for all algorithms described above, given an observation time of 60 seconds, where the patrol algorithm was unknown to the player. The bars represent the expected penetration detection ratio given the actual choices of the players' penetration spots. In order to compare the performance results obtained by the different algorithms, we used the Mann-Whitney U-test [57], which is a non-parametric test, suitable for data with no normal distribution (like the data in our case).

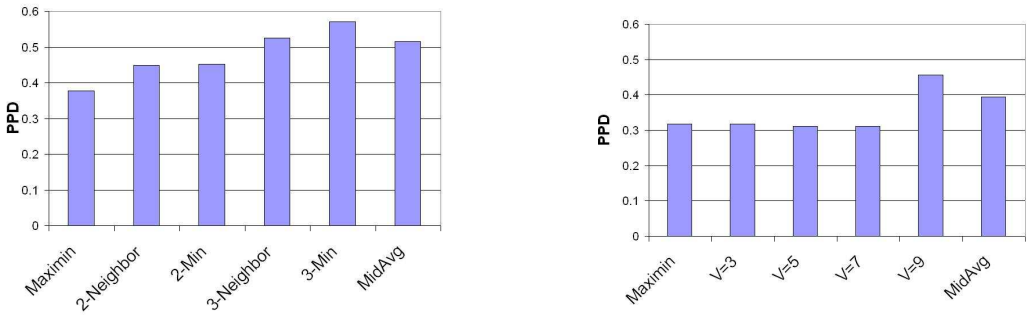


Figure 6.7: Results of the experiment for $d = 8, t = 6$ (on the left) and for $d = 16, t = 9$ (on the right), both for unknown p. The bars represent the expected penetration detection ratio of the robots given the choices of the players.

For the first case in which $d = 8, t = 6$ we can clearly see that the best-performing algorithm, i.e., the algorithm that achieved the highest expected probability of penetration detection based on the choices of the players, was v-Min for $v = 3$ (denoted by 3−min). Specifically, the results of 3−min were statistically significantly better than v-Neighbor and v-Min for $v = 2$ ($p-value = 0.001$ and $p-value = 0.01$, respectively), MaxiMin ($p-value = 0.003$) and MidAvg ($p - value < 0.002$). However, the results of 3−min were not significantly better than v-Neighbor for $v = 3$ (denoted by 3−neighbor).

In order to explain the advantage of using 3−min, we inspected the actual choices the players made concerning their penetration spots. Approximately 50% of the players decided to penetrate through one of the 3 segments with minimal ppd, and the expected ppd in these segments is 34%. In contrast, only approximately 29% of the players detected the weakest spot when executing the MaxiMin algorithm (in this case having an expected ppd of 24%). This means that the 3−min algorithm indeed had better predictions con-

cerning the penetration spots.

Another reason for the 3−min's good performance lies in the fact that the other 50% of the players who didn't choose to penetrate through the weakest spots, had better chances of getting caught by the 3−min algorithm also in the other segments. The MaxiMin algorithm attempts to strengthen the weakest spot, and thus it substantially decreases the probability of penetration detection in the other segments. For example, for $d = 8, t = 6$, the expected ppd in the non-weakest segments using the MaxiMin algorithm is 49%, whereas with the 3−min algorithm it is 83%. The minimal ppd, though, decreases from 24% to 11% with the 3−min algorithm.

Since the players did not obtain enough information to identify the exact weakest spots and enter through those spots, the use of the MaxiMin algorithm was not worthwhile. The 2−min and 2−neighbor algorithms suffer from the same problem, though not as profoundly as the MaxiMin does. Therefore they did not perform as well as the 3-min or 3-neighbor algorithms. The 3−min algorithm performed better than the 3−neighbor algorithm, yet not significantly better, since they both assume similar uncertainty level (3 segments).

Note that it may be worthwhile to enlarge the level of uncertainty, i.e., the v value in order to capture more choices of penetration spots. However, if $d = 8, t = 6$, for $v > 3$ the optimal algorithm is deterministic, which is highly predictable, and as shown in Section 6.2.1, it is easily manipulated by an adversary with even a small amount of information.

For $d = 16, t = 9$ we see from the expected ppd values that v-Min / v-Neighbor for $v = 9$ (denoted by $v - 9$) performs considerably better than algorithms with smaller v's (tested for $v = 3, 5, 7$), MaxiMin and MidAvg. However, we were not able to obtain statistical significance using the Mann-Whitney U-test. The reason lies in the fact that $v - 9$ expects the players to penetrate through one of the 9 weakest segments, and indeed 68% of the players chose to penetrate through these segments. Therefore the expected ppd consists of a large range of possible values that are not normally distributed, therefore even though the $v - 9$ produced highest levels of ppd, we could not show significance.

Analysis for known p:
Figure 6.8 describes the *expected* probability of penetration detection given the players' choice of penetration locations for $d = 8, t = 6$ and $d = 16, t = 9$ for all algorithms described above, given an observation time of 60 seconds,

where the patrol algorithm was disclosed to the player. The bars represent the expected penetration detection ratio given the actual choices of the players' penetration spots. In this case as well we used the Mann-Whitney U-test [57] for determining significance (since the data in our case as well has no normal distribution).

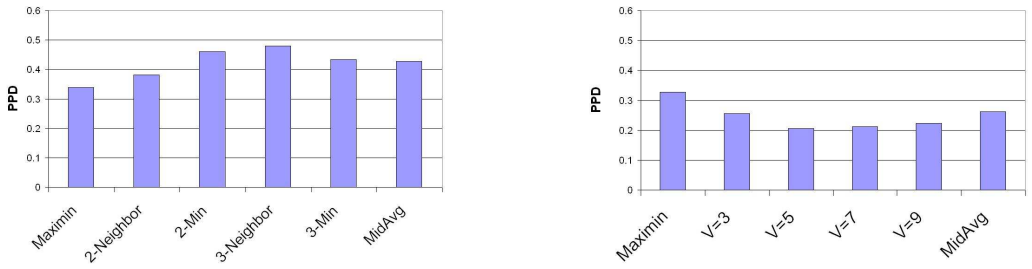


Figure 6.8: Results of the experiment for $d = 8, t = 6$ (on the left) and for $d = 16, t = 9$ (on the right), both when the patrol algorithm is presented to the player. The bars represent the expected penetration detection ratio of the robots given the choices of the players.

For the case in which $d = 16, t = 9$ the best-performing algorithm, i.e., the algorithm that achieved the highest expected probability of penetration detection based on the choices of the players, was the MaxiMin algorithm. Specifically, the results of MaxiMin were statistically significantly better than v-Neighbor v-Min for $v = 5, 7$ and 9 ($p - value = 0.009$, $p - value = 0.001$ and $p - value = 0.002$, respectively). However, we were not able to obtain significance results compared to MidAvg and $v - 3$. We assume that the reason for not finding significance lies in the fact that in these two cases the number of players that played these sets was less than 40. Specifically, 36 and 33 players played the MidAvg and $v-3$ sets, respectively, compared to 40 and 46 players that played the *maximin* and the $v - 9$ algorithms, respectively.

For the case in which $d = 8, t = 6$ we can see that the disclosure of the information regarding the patrol algorithm did not substantially change the performance of the different algorithms. Algorithm 3−min still outperforms the other algorithms, yet now (compared to the case in which the algorithm is unknown to the player) its performance is not statistically significantly better than any of the other algorithms (this fact is also clearly seen in the graph). Therefore the advantage of using the 3−min algorithm is not maintained here, and the difference between using the different algorithms

becomes relatively indistinct.

Another interesting discussion rises from examining the clear statistically significance results obtained in the first phase of the PenDet-Game (Section 6.2) to the difficulty in reaching statistically significance results in the second phase of the game. The reason for this lies in the fact that in the first phase, only three algorithms were compared - MaxiMin, Det and Combine. Each of these algorithms are substantially different from one another, and is it relatively easy to know them apart. For example in the first stage of the initial experiment, the difference in the expected ppd of each segment using the three different algorithms is tremendous (see Figure 5.4). Moreover, in the third stage of the initial experiment the probability p characterizing the patrol algorithm substantially changes between the algorithms. For example, if $d = 8$ and $t = 6$, $p = 0.7$ when using the MaxiMin algorithm vs. $p = 0.96$ when using Combine. When only little information was presented to the player (5 seconds in the second stage), the performance of the algorithms was not coherent and no significance results were obtained. The inability to prove significance in this case is originated in the fact that 5 seconds of observation does not give the player a chance to determine the patrol algorithm, specifically learn the nondeterministic nature of it. In the second phase of the experiment we tested 6 different algorithms for each setting, altogether 12 different algorithms. The difference between the algorithms (probability p characterizing it) are not always evident. For example, it would be difficult to differentiate between an algorithm with $p = 0.76$ (2−min) and one with $p = 0.77$ (2−neighbor). This provides an additional motivation for further examining the case of adversary having some knowledge on the patrolling robots, using new methods and/or new algorithms (see Chapter 9).

Part II

Additional Challenges in Multi-Robot Tasks

Chapter 7

Team Member Reallocation in Multi-Robot Formation

This chapter considers the task reallocation problem, where k robots are to be extracted from a coordinated group of N robots in order to perform a new task. The interaction between the team members and the cost associated with this interaction are represented by a directed weighted graph. Consider a group of N robots organized in a formation. The graph is the monitoring graph which represents the sensorial capabilities of the robots, i.e., which robot can sense the other and at what cost. The team member reallocation problem with which we deal is the extraction of k robots from the group in order to acquire a new target, while minimizing the cost of the interaction of the remaining group, i.e., the cost of sensing amongst the remaining robots. In general, the method proposed in our work shifts the utility from the team member itself to the interaction between the members, and calculates the reallocation according to this interaction cost. We found that this can be done optimally by a deterministic polynomial time algorithm under several constraints. The first constraint is that $k = \mathcal{O}(\log N)$. We show that other algorithms can be based upon this algorithm, and describe two of these algorithms that take under consideration more than one component in the utility function. The first algorithm handles prioritized components and the second handles weighted components. Lastly, we describe several other non-robotic domains in which this method is applicable.

7.1 Problem definition

The motivation for the following representation of the problem comes from the world of multi robot formation maintenance. In this domain, the robots monitor one another in order to maintain a formation. While in the formation, k of the robots are required to leave the team in order to acquire a new target. We wish to extract the robots in such a way that will minimize the monitoring cost of the original formation. A detailed description of the multi robot problem follows the general definition of the problem, which is applicable to other domains (as seen in Section 7.5).

Let $G = \{P_1, \ldots, P_N\}$ be a group of N homogenous team members. The interaction between team members can be represented by a cost function. Note that this cost function can be generalized into an interaction based *utility* function. An example of such a generalization is presented in section 7.3. The group has one root - a team member that acts as the leader. The set of members and the interaction between them can be presented as a directed

weighted graph $G = (V, E)$, where $v \in V$ are the team members, and the edges represent the interactions. Namely, $(v_i, v_j) \in E$ if an interaction exists between v_i and v_j with a $\text{cost}(v_1, v_2)$, which is the weight of the edge (v_i, v_j). An *Optimal Interaction Tree* (OIT) is built on this graph by simply running Dijkstra's shortest path algorithm between all vertices of the graph and the leader (similar to [51]). One main constraint is required on this graph as follows:

Constraint A: If $(v_1, v_2) \in E(\text{OIT})$ and $(v_2, v_3) \in E(\text{OIT})$, then $\text{cost}(v_1, v_2) + \text{cost}(v_2, v_3) < \text{cost}(v_1, v_3)$.

This means that the triangle inequality exists on the *optimal* tree of G, $\text{OIT}(G)$ (but not necessarily on G). Intuitively, if a path is chosen to be in $\text{OIT}(G)$, then there is no shorter path between the two edges of the path. Note that this does not mean that the triangle inequality holds for general edges of G, as depicted in Figure 7.1. In this example, the triangle inequality does not hold in G, for example $\text{cost}(a, c) + \text{cost}(c, e) \leq \text{cost}(a, e)$, thus clearly the edge (a, e) will not be in $\text{OIT}(G)$. The triangle inequality can still exist between some vertices, for example $\text{cost}(a, b) + \text{cost}(b, c) > \text{cost}(a, c)$.

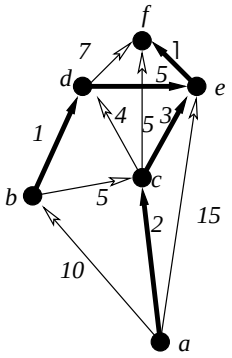


Figure 7.1: An example of a case in which the triangle inequality exists in $\text{OIT}(G)$, yet it does not exist in G.

The motivation behind this constraint is as follows. In order to achieve a relatively lower time complexity, we would like to restrict the options of the removal of nodes. Specifically, this constraint allows us to consider only leaves or subtrees of the tree. The constraint is necessary in order to avoid cases like the one depicted in Figure 7.2, in which the structure of G does not

follow Constraint A. In this case, illustrated in our robotic domain, the cost of the sensing robot r_1 by r_3 is 7, and lower than the cost of sensing r_1 by r_3 via robot r_2 ($10+10=20$). However, the edge (r_3, r_1) does not appear in the graph since r_2 lies exactly between r_3 and r_1, thereby blocking the sensing capabilities of r_3. This example conflicts with constraint A since edge (r_1, r_3) is not in the tree, whereas $\text{cost}(r_3, r_1) < \text{cost}(r_3, r_2) + \text{cost}(r_2, r_1)$, and indeed it would have been more profitable to remove r_2 and not r_3 which is the leaf.

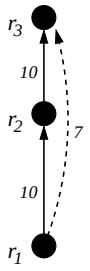


Figure 7.2: An example of a case where Constraint A is not fulfilled.

In many cases the leader of the team has unique characteristics that distinguish it from the other team members. Consequently there is no point in discussing the removal of the team leader. However, in some cases all robots are homogenous, including the leader, thus an additional property can be added to the problem, as given in the following definition.

Definition 1: In a graph $G = (V, E), V = \{v_1, \ldots, v_N\}$, a *potential leader* group $L = \{\tilde{v}_1, \ldots, \tilde{v}_M\}$, $1 \leq M \leq N$ is a subset of V containing possible leaders from the group. We denote the size of the potential leader group by M.

Having the OIT of the group, $k < N$ vertices should be extracted from the graph. The extraction should be done while satisfying the following basic objectives.

1. The cost of the remaining OIT should be minimal.
2. At least one of the vertices from the potential leader group of G should remain in the graph (this requirement is necessary due to the fact that we assume the remaining formation must have a leader, and only one of the potential leader group members can act as a leader).

It is assumed that all N team members can theoretically be extracted, hence if we are dealing with acquiring a new target or performing a new task, then all members are compatible for the mission (see an exception to this assumption in Section 7.2.3). Therefore, potentially, the number of different possibilities for extracting the k team members from the group is $\binom{N}{k}$. The algorithms described later in this chapter, which work in our settings and under our constraints substantially reduce the complexity to $\mathcal{O}(2^{k/2})$, which makes the algorithm polynomial if $k = \mathcal{O}(\log N)$.

Returning to the multi-robot domain, the root of the tree is the formation leader, the original graph is the monitoring multigraph where each vertex represents the location of a robot, and the edges represent the monitoring capabilities and cost of each robot of its peers. As proposed by [51], an Optimal Monitoring Tree, OMT, which is a special case of the OIT, is built on the monitoring multigraph. The OMT describes who each entity should monitor in order to minimize the cost of the sensing path from itself to the leader. The value of monitoring multigraphs and specifically OMTs, is its compatibility with real world scenarios, i.e., in the real world robots usually have limited sensing capabilities and the cost of sensing varies from one sensed object to another—depending on its distance and angle with respect to the sensing robot. When extracting k robots, the objective is to minimize the cost of sensing of the remaining group.

7.2 Team member reallocation focused on minimizing the cost of the remaining OIT

7.2.1 Team member reallocation with one possible leader

In this section we describe an algorithm that finds the optimal k vertices that should be extracted from the graph in order to minimize the cost of the remaining OIT. The Tree_Pruning algorithm described in this section, finds the optimal k vertices to be extracted, assuming that the leader cannot be changed.

The following lemma and its corollary provides the motivation behind the algorithm. The lemma proves that the algorithm should concentrate on removing the leaves or subtrees from the OIT rather than arbitrary vertices

without all vertices that are in the subtree rooted in that vertex. The reason lies in the fact that the OIT is built in an optimal manner, hence any removal of a vertex without all the subtrees rooted in it will force the vertices of those subtrees to find an alternative path towards the leader. Based on the optimality of the tree, this alternative either will incur the same cost or will be more expensive than the original one.

Lemma 22. *Consider an* $\mathsf{OIT}(G)$*, satisfying Constraint A. If vertex* v *that is not a leaf nor the leader is removed, then the sum of the weights of the edges of* $\mathsf{OIT}(G \setminus v)$ *will not decrease.*

Proof. In a DAG, every vertex that is not a leaf is an articulation vertex, meaning, removing it will disconnect the graph. Therefore all vertices connected to v should find another node to connect to, i.e., all $u_i \in V$ such that $(u_i, v) \in E$ and $(v, u) \in E$, should create a new edge (u_i, v_j) such that the cost of the DAG is minimized.

If $v_j = u$ then the proof is completed, since by Constraint A $\text{cost}(u_i, u) > \text{cost}(u_i, v) + \text{cost}(v, u)$. If $v_j \neq u$ then by minimality of the OIT it follows that if $\text{cost}(u_i, v_j) < \text{cost}(u_i, v)$ then the algorithm would have chosen u_i to point to v_j in the first place, contradicting the minimality of the OIT. □

Corollary 23. *In an* $\mathsf{OIT}(G)$ *satisfying Constraint A, the benefit gained from removing a leaf is greater than the benefit gained from removing one of its ancestors.*

The following definitions are used throughout the description of the algorithm.

Definition 2:

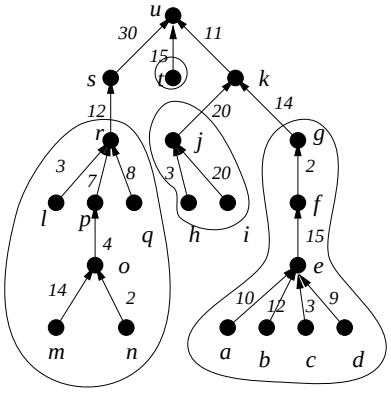
2.1 A *palindromic composition* of a number k is a collection of one or more positive integers whose sum is k. The number of palindromic compositions of a number k is $2^{\frac{k}{2}}$ [17].

2.2 A *bundle* originating in vertex v is the subtree rooted in vertex v. Vertex v *nests a bundle* of size t if the bundle originating in it has t vertices, including v.

2.3 In a directed tree $G = (V, E)$ where all paths are directed to the root of the tree, if $(v, u) \in E$ then u is called v's *pivot*.

From Corollary 23 we can assert that the gain from removing bundles or leaves from OIT(G) will be greater than the gain from removing arbitrary vertices from the graph. This fact motivates the algorithm Tree_Pruning, which exhaustively searches all possible combinations of leaves and bundles and picks the combination which results in the highest reduction in cost to the remaining OIT(G).

The Tree_Pruning algorithm (13) works as follows. First, it creates a table in which it stores the vertices in levels $1, \ldots, k$, where each level i, $1 \leq i \leq k$, contains vertices that nest a bundle of size i. For each of these elements it indicates the gain from removing the bundle originating in that vertex. This gain is simply the sum of all the costs of edges in this subtree, including the cost of the edge going from the root of the subtree to its pivot. After the table is created, the algorithm starts checking all palindromic compositions of the number k. For each composition $\alpha_1 + \alpha_2 + \ldots + \alpha_t$ the algorithm first checks whether it is feasible, i.e., whether there are components of sizes $\alpha_1, \ldots, \alpha_k$. If so - for each α_i it checks for the element with maximal gain in level α_i of the table. If the algorithm picks up non-disjoint bundles, then it checks the gain of removing each element of the non-disjoint set alone. Summing up the gains from removing the composition is compared to the current maximal gain, and the set resulting in the higher gain is saved. Finally, after all palindromic compositions of k are examined, the algorithm returns the set with the highest cost reduction upon its removal.



7	g		r			
6	f					
5	e					
4			p			
3		j	o			
2						
1	a b c d	h i	m n	l q	t	

Figure 7.3: An example of 7-bundling of a tree.

Theorem 24. *The* Tree_Pruning *algorithm finds the optimal k vertices to be removed from the group such that the cost of the remaining* OIT *is minimized.*

Algorithm 13 Algorithm $\mathsf{Team}_k = \mathsf{Tree\_Pruning}(G = (V, E), k)$

for each leaf $v \in V$ **do**
  Start building a k-bundle bottom-up:
  $C_{best} \leftarrow 0$ and $\mathsf{Team}_k \leftarrow \emptyset$.
  Add each subtree of size $1 \leq t \leq k$ to the table in row t and calculate its cost.
  Sort all elements in each row according to their cost.
  Generate a palindromic composition of the number k and sort each composition from left to right in decreasing order.
  for each possible composition C_j of $k = \alpha_1 + \alpha_2 + \ldots + \alpha_t$ **do**
    Check whether the composition is feasible.
    for each α_i in the composition, $i = 1, 2, \ldots, t$ **do**
      Pick highest order unmarked element from row t_i and mark it.
      if the elements are not disjoint, then check all possibilities: **then**
        Pick element with highest α_i, next don't pick it and pick element with next highest α_i and so on.
      Calculate the cost of the composition C_j.
      if $\text{cost}(C_j) \geq C_{best}$ **then**
        $C_{best} \leftarrow C_j$ and $\mathsf{Team}_k \leftarrow$ current composition.
Return Team_k

Proof. As seen in Corollary 23, the optimal benefit for the remaining tree is obtained by removing vertices that are not articulation points in the graph. Therefore the examination of all removal possibilities of leaves and bundles, as done by the algorithm, assures that the optimal group of k vertices will indeed be removed. □

Time Complexity: The time complexity of the preliminary work of building the table is $\mathcal{O}(N)$, as each vertex is visited once. The sorting of the rows will cost additional $\mathcal{O}(N \log N)$ time. The number of palindromic compositions of a number k is $2^{\frac{k}{2}}$ [17]. The algorithm might check each composition (the worst case) N times, if the chosen elements are not disjoint. Assuming that each approach to an element takes $\mathcal{O}(1)$ steps (depending on the data structure used), each composition is calculated in (the worst case) $\mathcal{O}(N)$ steps.

Therefore the total time complexity of the algorithm is $N \log N + N2^{\frac{k}{2}}$. Assuming that k is in order $\log N$, then the time complexity of the algorithm is $\mathcal{O}(N \log N + N\sqrt{N}) = \mathcal{O}(N^{1.5})$.

7.2.2 Team member reallocation with multiple possible leaders

Removing the leader v_{lead} can significantly decrease the total cost of the graph in cases where the weights of edges entering v_{lead} are considerably higher than the other weights in the graph. Therefore when examining the vertices, it can be highly profitable to examine removal of both leaves (or bundles) and the leader. The removal of the leader is possible only in cases where the potential leader group of the formation is of a size greater than one. Consider the case in which robots move in a formation. In this case, it is possible that several robots in the formation can lead the group. This is obvious in cases in which all robots are homogenous, whereby, theoretically, the number of robots in the potential leader group is N.

The order of extraction of the leading vertex is important, since changing the leader might result in a different structure of the OIT. In particular, it might result in changing the identity of the leaves and bundles. If we wish, for example, to extract three vertices from the graph, the result may vary if we pick a leaf, leader and a leaf from the new tree, as opposed to picking, say, two leaves and then the leader. Note that in some cases removal of a

leaf might result in changing the leader, depending on the leader election algorithm. However, in our case we assume that this does not happen.

In the extreme case in which $M = N$, the number of different possibilities for choosing k vertices - a combination of leaves and leaders, is $\mathcal{O}(2^k)$. See Figure 7.4 for an example.

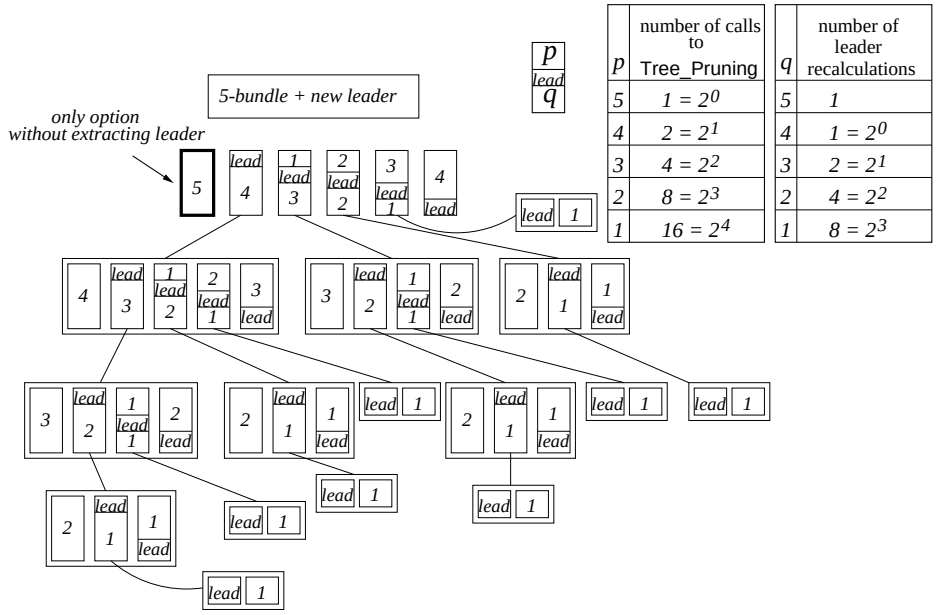


Figure 7.4: An example of the search tree of all possibilities of extracting $k = 5$ vertices from the graph where the leader can be elected as well in each step.

In the first level, $l = 1$, we can either pick k vertices using the Tree_Pruning algorithm (meaning, we do not change the leader), or change the leader first, second and so on until the k'th vertex is selected. Each of these options except the former branches out similarly in the next level ($l = 2$) where k decreases by one. If $M \geq k - 1$ then the formal time complexity analysis is as follows.

Assume that a leader is chosen in a round where t vertices remain to be chosen, $1 \leq t \leq k$. It is possible to pick p vertices before the leader is replaced and q afterwards, $0 \leq p, q \leq t - 1$ and $p + q = t - 1$. Therefore there are t different choices of the order in which the leader is extracted, in addition to the choice where the leader is not replaced. When a leader is replaced, the calculation of the p vertices chosen prior to it is obtained simply by running Tree_Pruning for p. The remaining q launch an additional level where, again,

they branch into $q+1$ new options. In order to calculate the complexity of finding the best allocation, we need to calculate the number of times each p appears (the q is calculated in the next level). As demonstrated in the table below, each number $i = 1, \ldots, k$ appears as p, i.e., above the leader line, 2^{k-i} times. Using Tree_Pruning, the complexity of each extraction of size i is $N \log N + N2^{\frac{i}{2}}$. Therefore the total complexity is $\sum_{i=1}^{k} 2^{k-i} \cdot (N \log N + 2^{\frac{i}{2}}) = \mathcal{O}(2^k N \log N) + N \sum_{i=1}^{k} 2^{k-\frac{i}{2}} = \mathcal{O}(2^k N \log N) + \mathcal{O}(2^k N) = \mathcal{O}(2^k N \log N)$. If $k = \mathcal{O}(\log N)$, then the complexity of choosing the members is $\mathcal{O}(N^2 \log N)$.

To this complexity we need to add the cost of recalculating the graph after a leader is extracted. As shown in [22], the complexity of calculating the OIT of a graph of size N given the identity of the leader vertex is simply running Dijkstra's shortest paths algorithm that takes $\mathcal{O}(N^3)$. If there are $\tilde{M}$ potential leaders, then the complexity is $\mathcal{O}(N^3 M)$. Hence here the time complexity can be bounded from above by the total number of qs, times $\mathcal{O}(N^3 M)$. The total number of qs, as demonstrated in Figure 7.4, is $\sum_{i=1}^{k-1} 2^{(k-1)-i}$, therefore the total time complexity is $N^3 M \sum_{i=1}^{k-1} 2^{(k-1)-i} = \mathcal{O}(N^3 M 2^k)$. Again, in our case $k = \mathcal{O}(\log N)$, hence the final complexity of the algorithm is $\mathcal{O}(N^4 M)$.

7.2.3 An algorithm variation, in which not all vertices can be extracted

In some cases, it is possible that there is a requirement that some team members cannot be extracted from the group. For example, if the robots moving in the formation are heterogenous, some might be incapable of performing the new task and thus cannot be chosen to do it. In another example, if the formation itself defines critical positions that cannot be deserted, it will dictate the identity of the robots that cannot be extracted. In both cases - driven by the old or new task requirements - some robots must remain in the original team. Converting it to our graph problem, if vertices should be extracted only from a subgroup G_1 of G, $G_1 \subseteq G$ and $|G_1| \geq k$, then a small variation of the basic algorithm can be used in some cases. First, if $|G_1|$ is exactly k, then the only option is that all vertices in G_1 be extracted.

Definition 3: In an OIT graph G, a vertex $v \in V(G)$ is called a *bundle blocker* if it cannot be extracted from the graph and the bundle above it stops spreading.

In our case, the bundle blockers are all vertices u_i such that $u_i \notin G_1$. If the bundle blockers lie in accumulating levels of depth k or more, then

a simple variation of Tree_Pruning can be used in order to find the optimal k vertices to be extracted. In this variation of Tree_Pruning, the algorithm should be run on the OIT graph G, but should stop at bundle blockers or at depth k, whichever comes first.

7.3 Multiple components in the utility function

When establishing the nature of the utility function, it is possible that more than one property will be taken into consideration. For example, one might want to consider both the cost of the remaining formation and the cost of the new formation. We consider two manners in which more than one consideration can be incorporated in the utility function: prioritized components and weighted components. We show examples in which the Tree_Pruning algorithm can be used in both cases.

Note that the ground rule which stands at the base of the use of the Tree_Pruning algorithm is that it is more profitable to remove leaves and bundles of the OIT. Hence any scenario in which we can incorporate the use of Tree_Pruning must apply to this rule. In the following examples we were motivated by the stability of the formation, whereby any change in the pivot of robots might cause instability of the formation. Therefore the following case suits the requirement that only bundles or leaves be removed, and thus the Tree_Pruning algorithm perfectly suits this problem.

7.3.1 Prioritized components

In prioritized components, the components are presented in a prioritized list, namely, component one is more important than component two and so on. Therefore we first maximize the utility according to the first priority. In case of a tie, we examine the component listed second in the priorities, and if that results in a tie we move on to the third priority component and so on.

A good example of this would be when we wish to minimize events that might undermine the remaining group's formation stability as well as minimize the monitoring cost. For instance, we assume that robots leaving the formation in order to acquire a new target continue in a straight line from their current location towards the new target. First, we would like to cause minimal changes to the current OIT, so that robots will not have to switch

pivots and thereby cause temporary instability. Thus only leaves or bundles should be removed, and the leader should remain intact. Second, we wish to minimize collisions between the robots leaving for the new target and the ones remaining in the formation. As seen in Figure 7.5, if v_2 moves straight towards t_G and maintains its predefined velocity, it will intersect with v_3. In addition, if v_5 moves towards t_G then at some point it will block v_4's view of its pivot, v_3. Third, we wish to minimize the monitoring cost of the remaining formation. Hence the prioritized components list is as follows.

1. Minimize collisions between the robots leaving for the new target and the ones remaining in the formation
2. Minimize the incidents of robots leaving the formation while, at some point, crossing an OIT edge, thus hiding the pivot of some robot remaining in the formation and potentially causing it to divert from the group formation.
3. Minimize the remaining group's monitoring cost.

The Prioritized_Pruning algorithm works as follows. First, assuming that the robots are homogeneous, it is simple to calculate the expected intersections between paths of robots leaving the formation and the remaining OIT vertices (robots) and edges. For each robot r_i (vertex v_i) that leaves towards goal point p_G, the algorithm checks against all robots $r_j \neq r_i, r_j \neq r_{lead}$ whether the path of r_i hides the outgoing edge from robot r_j (vertex v_j). If so, it adds t_j to the entry of r_i in a pre-specified Table under column E (see for example Figure 7.5). If the robots themselves intersect, then r_j is added to Table under column I. After creating this table, Tree_Pruning is run on the graph where three features are examined in each step: I intersections, E intersections which are extracted from the Table, and the cost incurred by the remaining OIT (in this order).

The Prioritized_Pruning algorithm is guaranteed to find the k robots that will minimize the potential disturbance to the formation satisfying the criterions we defined. The time complexity of the algorithm is composed of two steps. In stage 1, a simple brute force algorithm that finds the intersections will take $\mathcal{O}(N^2)$ steps by simply comparing each pair of robots. Stage 2 is similar to the Tree_Pruning algorithm with an addition of maximum $\mathcal{O}(k)$ comparisons at each step, hence the complexity is $\mathcal{O}(N^{1.5} \log N)$ (assuming that $k = \mathcal{O}(\log N)$), and altogether the complexity is $\mathcal{O}(N^2)$.

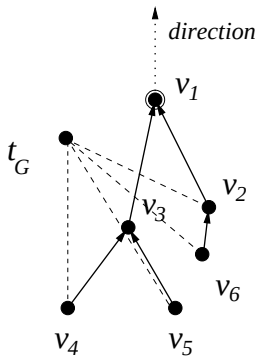


Table:

	E intersection	*I intersection*
v_2	Ø	v_3
v_3	Ø	Ø
v_4	Ø	Ø
v_5	v_4	Ø
v_6	v_4 v_5	Ø

Figure 7.5: An example of a path/edge intersection.

Algorithm 14 Algorithm $\mathsf{Team}_k = \mathsf{Prioritized\_Pruning}(G = (V, E), k, t_G)$

1: **for** each vertex $v_i \in V$ such that v_i is not the leader **do**
2: Go over all vertices of the graph except for vertices in the bundle originating in v_i.
3: **if** the outgoing edge of vertex v_j intersects the path of v_i on its course towards t_G **then**
4: Add v_j to $\mathsf{Table}(v_i, E)$.
5: **if** v_i on its course towards t_G intersects v_j **then**
6: Add v_j to $\mathsf{Table}(v_i, I)$.
7: Run procedure $\mathsf{Tree\_Pruning}(G, k)$ with the following modifications.
8: Set $E_{best} \leftarrow \infty$, $I_{best} \leftarrow \infty$ and $\mathsf{Team}_k \leftarrow \emptyset$.
9: Check the number of E and I intersections between members of the current chosen elements and the remaining ones and store them in E_{cur} and I_{cur}, respectively.
10: **if** $I_{cur} < I_{best}$ **then**
11: $I_{best} \leftarrow I_{cur}$ and
$\mathsf{Team}_k \leftarrow$ current composition.
12: **if** $I_{cur} = I_{best}$ and $E_{cur} < E_{best}$ **then**
13: $I_{best} \leftarrow I_{cur}$ and $\mathsf{Team}_k \leftarrow$ current composition.
14: **if** $I_{cur} = I_{best}$ and $E_{cur} = E_{best}$ **then**
15: Check the difference between the cost of the composition and save the best of two choices, as done in $\mathsf{Tree\_Pruning}$.
16: Return Team_k.

7.3.2 Weighted components

Formally, weighted components allow us to assign an accumulating percentage for each component, and choose the option resulting in the maximized utility value U. Each of these utility values is composed of l different components $\{u_1, \ldots, u_l\}$, and w_t is the weight of the utility component u_t, where $\sum_{t=1}^{l} w_t = 1$. Therefore the weighted utility value U is calculated as: $U = \sum_{t=1}^{l} w_t^i u_t^i$

In the following example we assume that robots leaving the formation remain in their current location, i.e., they will change their relative position to other robots. Our objective is to minimize both the remaining group's monitoring cost and the monitoring cost of the robots leaving the formation, according to their location at the moment they leave the formation. This is applicable in cases where a subgroup of robots is required to leave the formation and create a formation of its own with minimal sensorial cost of the new formation. Our original assumption is that we wish to minimize disruptions of the original formation, thus we will examine the removal of only leaves and bundles, and leave the leader intact. This property will allow us to use the Tree_Pruning algorithm in this case. Note that if we consider only the monitoring cost of the leaving robots, we might have needed to remove vertices that are not necessarily leaves or bundles. For example, if $k = N - 2$, then the optimal choice of removal would be the removal of the minimal edge in the OIT, regardless of its location if only the monitoring cost of the extracted group is considered. Therefore, as we have shown in the previous subsection, the initial assumption that we remove only leaves and bundles is crucial for the use of the Tree_Pruning algorithm and ensures its complexity.

Minimizing the remaining group's sensorial cost (first component) contradicts, in most cases, the minimization of the extracted group's sensorial cost. This contradiction is exemplified in Figure 7.6, and results straightforward from the fact that the first minimization requirement would cause the removal of the most expensive edges, while the second component would require the removal of the least expensive edges from the graph.

Use w_1 to denote the weight of the utility component of the remaining formation's monitoring cost , and w_2 to denote the weight of the utility component of the extracted formation's monitoring cost. The Weighted_Pruning algorithm, then, works as follows. For each possible choice of k robots, the algorithm calculates the cost of the remaining OIT multiplied by w_1, the cost

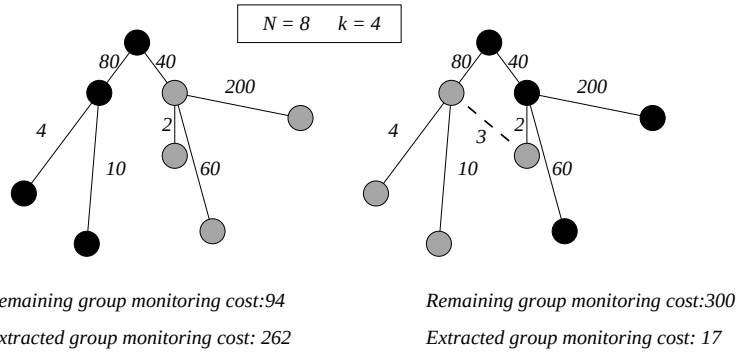


Figure 7.6: An example of a contradiction between the two utility components: the remaining group's OIT cost and the extracted group's OIT cost. Here $N = 8, k = 4$ and the optimal choice of nodes for removal is colored in gray, while the remaining nodes are colored in black.

of the extracted OIT multiplied by w_2, and it sums the two values. If the resulting value is lower than the lowest value obtained so far, this is saved as the potential optimal choice. After all choices have been checked, the choice with the optimal utility is reported.

Algorithm 15 Algorithm $\mathsf{Team}_k = \mathsf{Weighted\_Pruning}(G = (V, E), k, w_1, w_2)$

1: Run procedure $\mathsf{Tree\_Pruning}(G, k)$ with the following modifications.
2: Set $E_{best} \leftarrow \infty$, $I_{best} \leftarrow \infty$ and $\mathsf{Team}_k \leftarrow \emptyset$.
3: $C_j \leftarrow$ current composition.
4: $C_R \leftarrow$ cost of $\mathsf{OIT}(G \setminus C_j)$, and $C_E \leftarrow$ cost of $\mathsf{OIT}(C_j)$
5: $C_{cur} \leftarrow w_1 \times C_R + w_2 \times C_E$.
6: **if** $C_{cur} < C_{best}$ **then**
7: $\quad C_{best} \leftarrow C_{cur}$ and $\mathsf{Team}_k \leftarrow C_j$.
8: Return Team_k.

Algorithm Weighted_Pruning is guaranteed to find the k robots that will minimize the total utility according to the weights we received from the user. The time complexity of the algorithm is identical to the time complexity of the Tree_Pruning algorithm, since we go over the entire graph only once for each possible composition,, which leaves us with a time complexity of $N2^{\frac{k}{2}}$ needed to check all compositions.

7.4 Empirical Evaluation

We implemented the three algorithms described herein, Tree_Pruning, Prioritized_Pruning and Weighted_Pruning, in order to perform an empirical evaluation of the algorithms. The implementation was done using the Player/Stage simulation package [44], a practical and popular development tool for both simulated and real robots.

We simulated 16 robots, traveling in one of three formations commonly tested in general multi-robot formation problems (e.g. [51]) - see Figure 7.7:

A. Diamond

B. Triangle

C. Arrowhead

The edges between the robots in each formation were given weights according to the cost of sensing, similar to the weights given in [51]. In the first step, the robots built a spanning tree, instructing each robot which other robot to monitor in order to minimize the cost of sensing inside the formation. The Spanning trees are indicated by bold arcs in Figure 7.7.

Following the first phase of constructing the minimal spanning tree, we executed the three algorithms on the formation: Tree_Pruning, Prioritized_Pruning and Weighted_Pruning. We were interested in revealing which robots were extracted as the output of each algorithm, or more specifically what different outputs would be returned by each algorithm for the same formation. We wanted to test whether the Prioritized_Pruning algorithm would indeed answer possible problems raised by the use of Tree_Pruning in the multi-robot formation domain.

We continue with a description of the results of the algorithms' performance on the chosen formations.

First, the robots executed Tree_Pruning in order to reallocate 4 out of the 16 team members to a new task. The extracted robots were instructed to remain in their position while the remaining formation continued their movement in their initial direction. We obtained the following results. In formation A, robots $11, 13, 15$ and 16 were extracted from the team (Figure 7.8). In formation C, robots $13, 14, 15$ and 16 were extracted (Figure 7.8). The more interesting results were obtained in formation B, where robots $8, 9, 12$ and 14 were extracted. Since the extracted robots remained in place

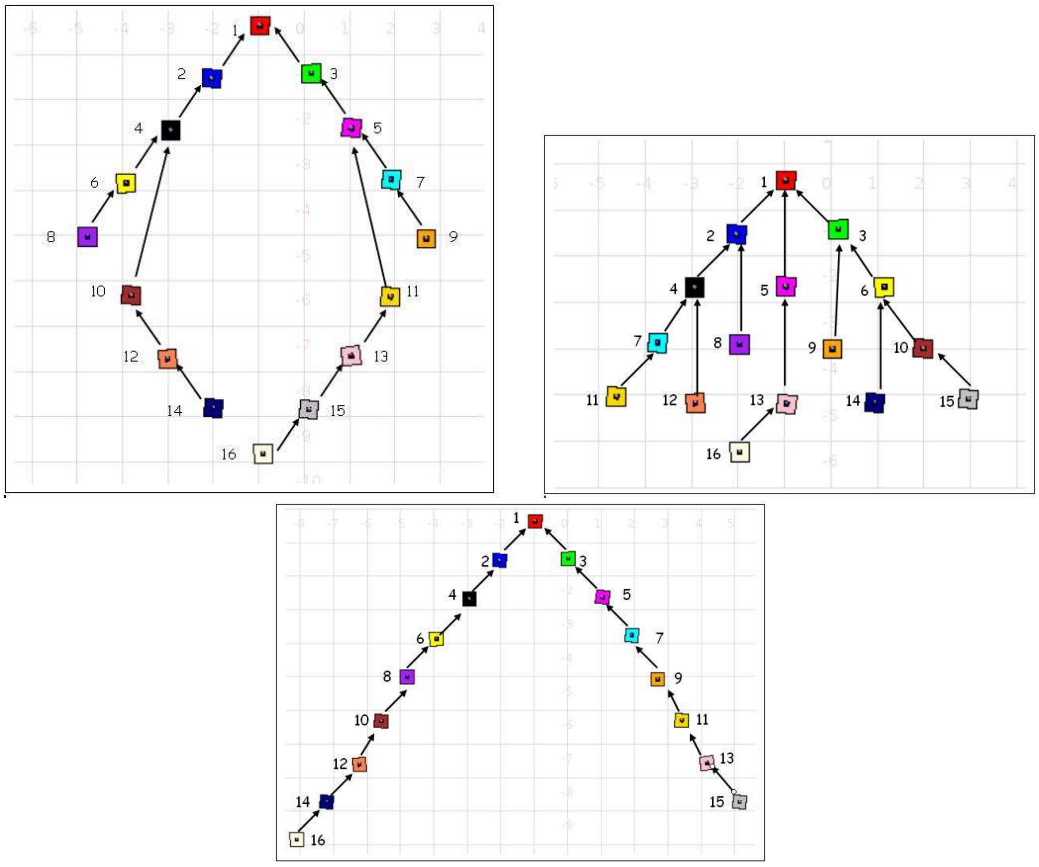


Figure 7.7: Three different formations tested in our Player/Stage simulation. The arcs represent the edges of the minimal sensing tree from all robots to the leader (robot 1).

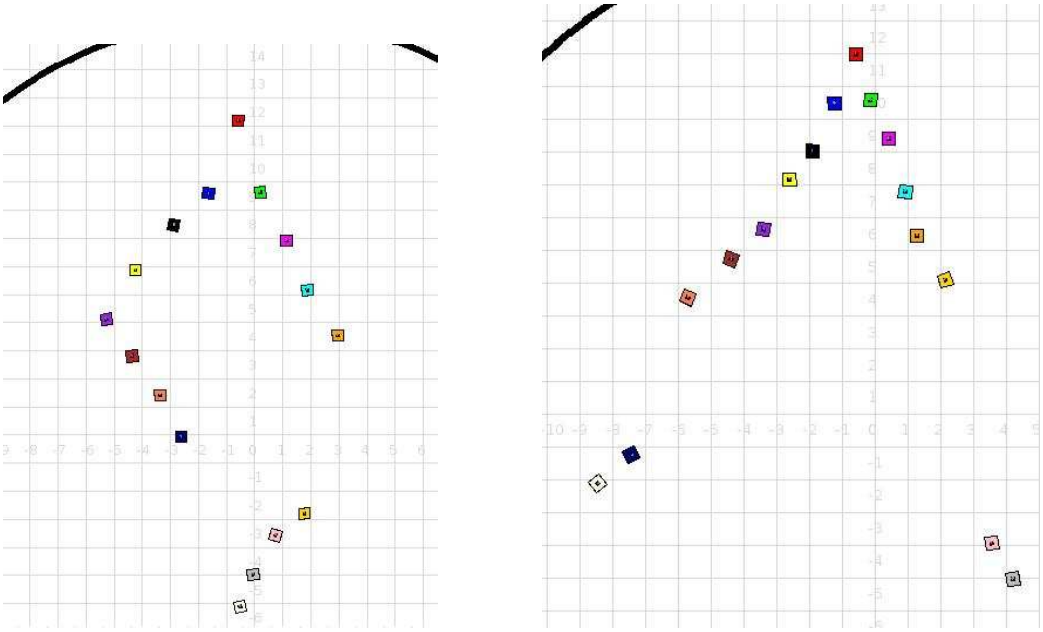


Figure 7.8: Execution of the Tree_Pruning algorithm on formation A (left) and C (right). The snapshot was taken approximately 15 seconds after the extraction.

and the remaining formation continued straight, robot 8 collided with robot 16 (see Figure 7.9).

The choice of robots to be extracted in formation B using Tree_Pruning strengthens the motivation to use the Prioritized_Pruning algorithm. Indeed, when executing the Prioritized_Pruning algorithm for formation B, the extracted set of robots was $9, 11, 14$ and 15. In this case, the target point of the robots was behind the formation, simulating a similar behavior given to algorithm Tree_Pruning, and thus emphasizing how the Prioritized_Pruning algorithm is able to solve the problem evolving from the use of the Tree_Pruning algorithm (see Figure 7.9).

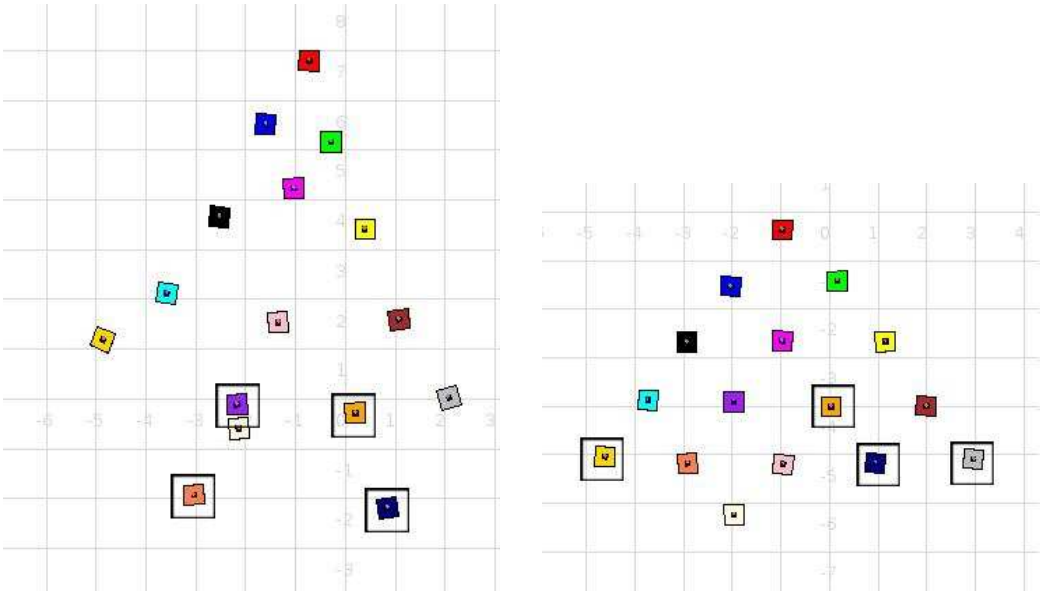


Figure 7.9: Execution of the Tree_Pruning (left) and the Prioritized_Pruning (right) algorithms on formation B. The extracted robots are denoted by a surrounding square.

When we used the Weighted_Pruning algorithm with $w = 0.5$, the extracted robots were $11, 13, 15$ and 16 in formation A, which is the same set that was extracted by Tree_Pruning. However, in formations B and C when adding the consideration of the weight of the extracted team of robots, the set of extracted robots was different. In formation B the set of extracted robots was $4, 7, 11, 12$, and in formation C the set was $10, 12, 14$ and 16. We checked the output also given other weights ($w = 0.2$ and $w = 0.8$), however in both

formations the resulted chosen team was similar. This can be explained by the fact that the weight of edges between the robots that cannot sense one another is ∞. Consequently any choice of a set of robots that would be extracted such that even one robot would remain disconnected from the other team members would receive a weight of ∞, and therefore that set would not chosen.

7.5 Applicability in additional domains

The general representation of the problem as *team member* reallocation, makes it applicable in other domains, in addition to the Multi-Robot Task Allocation (MRTA) domain, which motivated the current study.

One example is a variation of the ***dependency tree*** [75]. A dependency tree $G = (V, E)$ describes a group of N tasks (vertices) with prerequisite relation, i.e., an edge $(u, v) \in E$ exists if u has to be executed before v, and $\text{cost}(u, v)$ is the cost of executing v after u. The root of the tree is, then, the task that has to be executed last. In our case, we use a slight variation of the dependency tree. Here, we are given one task that should be conducted last and the interaction between all other tasks. If two tasks v_1 and v_2 are independent, then if v_1 is executed before or after v_2 their cost will be the same. If v_1 and v_2 are dependent, then without loss of generality, v_1 can rely on the fact that v_2 will perform a part of its task, thus $\text{cost}(v_1, v_2)$ in this case will be smaller than the cost in the independent case. The OIT describes the optimal tree of execution of the tasks. The requirement is to remove k tasks from the group such that the cost of the remaining execution tree is minimized.

The ***warehouse assembling*** problem presents an additional example in which the OIT is applicable. In the warehouse assembling problem we are given a set of N warehouses located in N distinct positions, and all the trucks are heading towards one main warehouse. The vertices of the graph represent the warehouses, and the edges represent the distances between two warehouses (note that triangle inequality does not apply). The objective is for any number of trucks to visit all warehouses in minimal time. The OIT represents the optimal tree of paths from all warehouses to the main warehouse. The number of trucks t is, then, the number of leaves in the OIT. The requirement is to close k warehouses in order to cut back expenses while not increasing the value t, and thus remain with the assembling tree with

the lowest cost.

The last problem is the ***network broadcast*** problem, in which we are given a network with one source vertex that should constantly broadcast messages to the rest of the network. The edges represent the cost of the link between every two vertices, and the OIT is the optimal broadcast tree. Once again in this case we are required to remove k vertices in order to cut back expenses and remain with a broadcast tree with the lowest cost.

Chapter 8

Improving Efficiency in Multi-Robot Area Coverage

In this chapter, we discuss the problem of building efficient coverage paths for a team of robots. An efficient multi-robot coverage algorithm should result in a coverage path for every robot, such that the union of all paths generates a full coverage of the terrain and the total coverage time is minimized. A method underlying several coverage algorithms, suggests the use of spanning trees as a base for creating coverage paths. However, overall performance of the coverage is heavily dependent on the given spanning tree. This work focuses on the challenge of constructing a coverage spanning tree for offline coverage that minimizes the time needed to complete coverage. Our general approach involves building a spanning tree by growing sub-trees from the initial location of the robots. We first provide a sound discussion and results concerning the construction of the tree as a crucial base for any efficient coverage algorithm. We then describe a polynomial time tree-construction algorithm for coverage. With extensive simulations we show that the use of this algorithm significantly improves the coverage time of the terrain even when used as a basis for a simple, inefficient, coverage algorithm. In addition, the solutions we propose in this work guarantee robustness to failing robots: the trees are used as a base for robust multi-robot coverage algorithms.

8.1 Motivation for building new spanning trees

In this section we describe the motivation behind our construction scheme of the trees. First, we show that the structure of the spanning tree has a crucial role in the coverage time obtained by algorithms that use the tree as a base for coverage. We prove that any coverage algorithm, even an optimal one, cannot achieve as low coverage time as can be achieved by using a different tree. Second, we show that a spanning tree, which by itself obtains the optimal coverage time, does not necessarily exist, hence the theoretical optimal coverage time might remain unreachable in some cases. Finally, we describe our definition of *optimal* spanning trees and explain the rationale behind this definition.

8.1.1 Importance of the spanning tree structure

An optimal time coverage algorithm for a system with k robots will (theoretically) result in total coverage time of $\lceil \frac{N}{k} \rceil$. Even the most basic multi-robot

coverage algorithm will result in such a coverage time if the robots are uniformly placed along the spanning tree path, i.e., within distance of at most $\lceil \frac{N}{k} \rceil$ from one another.

We argue that the choice of the spanning tree has crucial consequences on the coverage time obtained by algorithms using the spanning tree as a base for coverage. This is more evidently seen in algorithms that do not diverge from the spanning tree path, such as the MSTC algorithms. Consider, for example, the Opt_MSTC algorithm. These algorithms create optimal paths along the spanning tree for k robots, not allowing (nonfaulty) robots to bypass one another during the execution of the coverage algorithm. In this case, even in the worst initial distribution case in which all robots are bundled in their initial position, the best possible improvement will result in an improvement factor of approximately 2 : from $N - k + 1$ to $\frac{N-k}{2} + 1$. On the other hand, the improvement by spreading the robots along the spanning tree can reach nearly a factor of k: from $N - k + 1$ to $\frac{N}{k}$.

An illustration of the importance of the right choice of the spanning tree is given in Figure 8.1. The figure presents an example of a terrain in which $N = 36$, $k = 3$ and two different trees are suggested as a base for coverage. The spanning tree is described by the bold lines, and we use the different kinds of dashed lines to describe the spanning tree path, whereby each dashed line represents the distance between two adjacent robots along the path. In order to clarify the example, the section between each two adjacent robots is given a different background as well. Note that in both grids the robots are initially located in the same positions. The tree in Figure 8.1a places the robots uniformly along the tree path, thus a coverage time of $\lceil \frac{N}{k} \rceil$ is easily obtained if the robots simply follow the tree path in a counterclockwise direction. However, in Figure 8.1b. the robots are placed arbitrarily along the tree path, thus any multi-robot coverage algorithm, based on the spanning tree, will find it hard to meet such coverage time.

A formal statement regarding the possible improvement in coverage time obtained by algorithms vs. improvement obtained by changing the tree is given in Theorem 25. First, let us introduce the following definition. Note that we distinguish between a *procedure* that is executed on the input to generate the tree, and an *algorithm* which is the coverage algorithm executed given the input tree.

Definition: Given the initial positions of k robots on a terrain with N cells, let M be the coverage time of the terrain obtained by the basic NB_MSTC

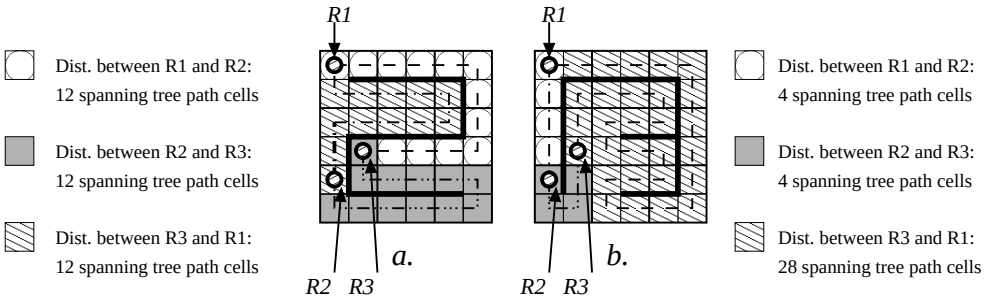


Figure 8.1: An illustration depicting how different trees can influence coverage time.

algorithm. A procedure $\mathcal{P}$ or an algorithm $\mathcal{A}$ are said to ensure an *improvement factor* t, if the coverage time obtained by NB_MSTC after applying $\mathcal{P}$ to the input, or the coverage time obtained by $\mathcal{A}$ for the same input is $\frac{M}{t}$.

Theorem 25. *Any multi-robot coverage algorithm for homogenous robots based on a spanning tree which does not divert from the spanning tree path will result in a maximal improvement factor of at most* 2.

Proof. Denote the distance between the initial location of robot R_i and R_{i+1} on the spanning tree path by D_i (also known as the segment D_i), and let $D_{max} = \max_{1 \leq i \leq N}\{D_i\}$. Clearly, the coverage time obtained by NB_MSTC is exactly D_{max}. Also, D_{max} determines the coverage time of any coverage algorithm $\mathcal{A}$ that does not divert from the spanning tree path. As the robots are homogenous and cannot bypass one another (assuming they are non-faulty), an improvement in the coverage algorithm can reduce from D_{max} to $\lceil \frac{D_{max}}{2} \rceil$ if robots on the extremity of D_{max} would simply walk towards one another while covering the terrain. If there is some other segment D_j which requires coverage time of some $t' > \lceil \frac{D_{max}}{2} \rceil$, then the new coverage time is t'. Note that t' can be smaller than the distance D_j if an algorithm allowing backtracking is permitted. In other words, the improvement factor is

$$\frac{D_{max}}{\max\{\lceil \frac{D_{max}}{2} \rceil, t'\}} \leq 2$$

□

While the change of the coverage algorithm can result in an improvement factor of at most 2, the example described in Figure 8.2 leads us to the

conjecture that an improvement factor due to a change in the tree can reach almost the value of k. As seen in Figure 8.2b, the coverage time obtained by NB_MSTC is $N - k = 56 - 3 = 53$, while the coverage time obtained by the same algorithm on a spanning tree constructed in a way that places the robots in an equally scattered manner along the tree (Figure 8.2a.) is $\lceil \frac{N-k}{k} \rceil = 19$, hence the improvement factor obtained by changing the tree is $\frac{53}{19} \approx 2.8$, which is almost k.

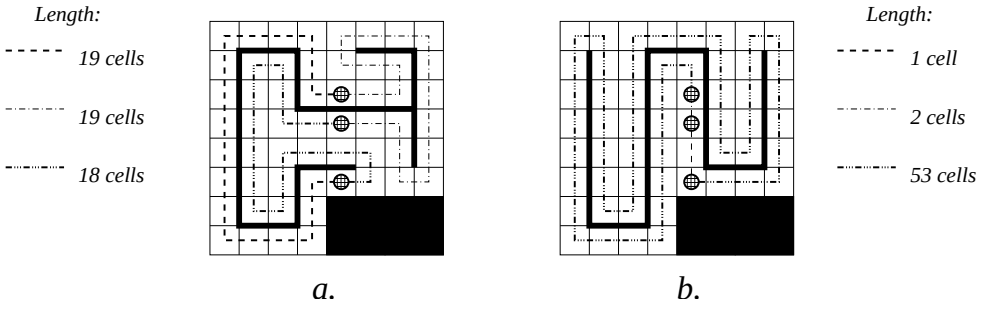


Figure 8.2: An example of a case in which the improvement factor is almost k if the tree is appropriately constructed.

We have established the fact that the choice of a spanning tree can have far reaching consequences on the coverage time of the terrain, possibly more than the choice of the coverage algorithm. Moreover, a spanning tree that places all robots within a distance of at most $\lceil \frac{N}{k} \rceil$ will, by itself, result in the optimal coverage time. Unfortunately, such a tree does not necessarily exist. For example, in Figure 8.3, $N = 16$, $k = 2$ and all possible spanning trees are described. The minimal maximal distance between two consecutive robots over all possible spanning trees is 10 cells, where $\lceil \frac{N}{k} \rceil = \lceil \frac{16}{2} \rceil = 8$.

In our tree construction scheme we will try to approximate this optimal dispersion of robots along the spanning tree. We will do so by trying to satisfy the following objective, as much as possible. First, let $\tilde{G}$ be a grid with $N/4$ cells, possibly containing obstacles (the obstacles are not counted as cells). Let G be $\tilde{G}$'s fine grid after dividing each cell into four cells of size D.

Objective: Given the initial locations of k robots in cells of G, find a spanning tree of $\tilde{G}$ that minimizes the maximal distance between every two consecutive robots along the spanning tree path.

The idea behind this objective is that it spreads the robots as uniformly

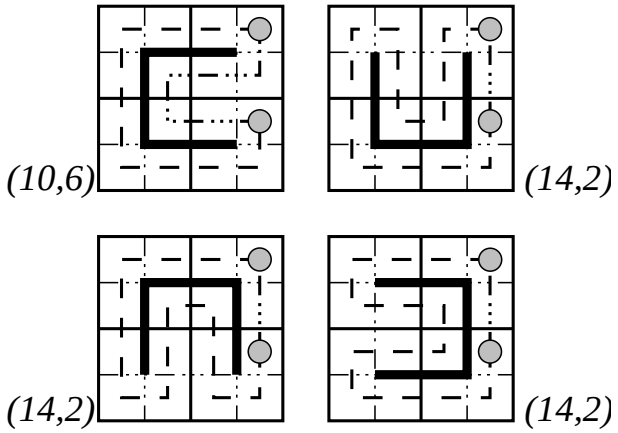


Figure 8.3: An example of a case in which there is no spanning tree that has maximal distance of $\lceil \frac{N}{k} \rceil = \lceil \frac{16}{2} \rceil = 8$ between consecutive robots along the spanning tree path. The numbers in parenthesis describe the distance between two robots along the spanning tree path.

as possible along the spanning tree path. The construction of an *optimal* tree, that will achieve accurately the objective, is believed to be $\mathcal{NP}$-hard [88]. Hence our tree construction algorithm can be considered a heuristic algorithm for the problem of finding the optimal tree for the coverage task.

8.2 Tree construction algorithm

In this section we describe a spanning tree construction algorithm, Create_Tree. This algorithm creates spanning trees while considering the initial location of all robots in the team and the objective described above, i.e., it tries to minimize the maximal distance between any two adjacent robots on the tree.

The general algorithm, described in Algorithm 16, is composed of two stages. In the first stage, a subtree is created gradually for each robot starting from the initial position of the robot, such that in each cycle either one or two cells are added to each subtree. Denote the subtree originated in R_i by T_{R_i}. The cells are chosen in such a way that maximizes the distance from the current expansion of all other trees. The algorithm tries to find the longest possible path for the tree. When it fails to continue, it tries to perform

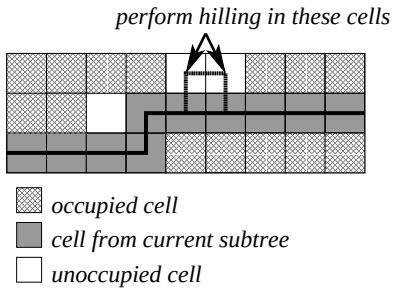


Figure 8.4: An illustration of the Hilling procedure.

Hilling, whereby it looks for ways to "stretch" the path as follows. It searches for two joint unoccupied cells adjacent to the path. If it finds such cells, it adds them to the path as demonstrated in Figure 8.4. If the algorithm fails to find more hills, then it expands the tree, from both sides of the path, in a BFS (breadth-first-search) manner. It first attempts to add one cell near the origin of the tree (initial position of the robot), then it checks for a possible free adjacent cell of its sons, and so on, until the entire grid is covered by all k disjoint subtrees.

In the second stage of the algorithm, after the k subtrees have already been generated they only have to be connected(second stage). Denote an edge connecting two different trees T_{R_i} and T_{R_j} by $\mathrm{br}(T_{R_i}, T_{R_j})$. As we are given k subtrees that need to be connected to one tree covering the entire grid, we need to find $k-1$ bridges. These bridges should be chosen in such a way that the resulting tree does not contain cycles or, equivalently, cover the entire grid. For example, if $k = 4$ then a possible valid choice of bridges are
$\{\mathrm{br}(T_{R_1}, T_{R_2}), \mathrm{br}(T_{R_1}, T_{R_3}), \mathrm{br}(T_{R_3}, T_{R_4})\}$, where
$\{\mathrm{br}(T_{R_1}, T_{R_2}), \mathrm{br}(T_{R_2}, T_{R_3}), \mathrm{br}(T_{R_1}, T_{R_3})\}$ is invalid, as T_{R_4} remains disconnected. Create_Tree randomly picks a valid choice of $k-1$ bridges, and calculates the maximal distance between two adjacent robots on the tree according to the **fine** grid. It repeats the process k^α times, and reports the best tree it observed, according to the above criterion. The value of α is chosen empirically.

Clearly, the algorithm provides complete coverage of the terrain, as the first stage of constructing subtrees does not end before every cell is occupied by some subtree. The first stage terminates, since in each cycle at least one cell is added to at least one subtree, and given a finite terrain the algorithm

halts. A formal proof of the completeness of Create_Tree is given in Lemma 26.

Algorithm 16 Procedure Create_Tree

```
1: Build k subtrees as follows.
2: for every robot R_i, 1 ≤ i ≤ k do
3:    for each possible next cell (up, down, right, left) do
4:       Compute the Manhattan distance from the current location of all
         other robots.
5:    if more than one possible next move exists then
6:       pick the one whose minimal distance to any other robot is maxi-
         mized.
7:    if there is no next possible move then
8:       perform Procedure Hilling from the last main branch.
9:    if failed to find an unoccupied cell in Hilling then
10:      Branch-out in a BFS manner from the main branch
11: Find all possible bridges between the k trees.
12: for i = 0 to max{k^α, N} do
13:    At random, find a valid set of bridges B_i between trees such that they
       create one tree with all N vertices.
14:    Compute the set S_i of distances between every two consecutive robots
       on the tree.
15:    Best_Result is initialized with S_0.
16:    if the maximal value in S_i is lower than the maximal value in
       Best_Result then
17:       Best_Result ← S_i.
18: Return the tree associated with Best_Result.
```

Lemma 26. *Procedure* Create_Tree *generates a tree that spans the entire graph G.*

Proof. Assume, for contradiction, that one cell C that is not covered by any subtree T_i, $1 \leq i \leq k$ exists. Since the map is finite, then some cell C‘ adjacent to C that is connected to a sub-tree originating in the initial location of some robot R_a exists. If C was not covered by the algorithm, then T_a must have finished all its phases - initial phase, i.e. Hilling and branching out. But if, while branching out, C' was traversed and C was empty, then it would have added C to T_a, leading to a contradiction. If both C and C' are empty,

then some $C'' \in T_b$, exists such that C is adjacent to C''. Similarly, either we have a contradiction, or C'' also is not in T_b. This continues until we find that all cells are not in any tree, or all cells are in the tree (by contradiction). The former case is impossible, as at least the initial location of a robot belongs to its subtree, hence we are done. □

Theorem 27. *The time complexity of the* Create_Tree *algorithm is* $\mathcal{O}(N^2 + k^\alpha N)$.

Proof. In the stage where k subtrees are created, in the worst case when adding one cell to a subtree the algorithm runs over all current cells in the subtree (during Hilling or while branching out), hence the complexity is at most $\mathcal{O}(N^2)$. In the second stage, where the trees are connected, k^α different choices of trees are examined, each time the entire tree is traversed, thus the complexity of this stage is $\mathcal{O}(k^\alpha N)$. Hence the entire complexity of the algorithm is $\mathcal{O}(N^2 + k^\alpha N)$. If the distance measure is the shortest paths, then calculating all-pairs shortest paths is $\mathcal{O}(N^3)$ ([23]). □

8.2.1 Using different distance measures

The Create_Tree Procedure first creates k subtrees, and then connects them. The process of constructing the k subtrees is done while spreading each tree *away* from the other trees. The distance measure used to determine how distant the trees are was initially simply the Manhattan distance ([2]). In this work, we used three different distance measures: Manhattan distance, Euclidean distance and Shortest paths (following Floyd's all-pairs shortest paths algorithm [23]). Note that the time complexity of the shortest paths algorithm is $\mathcal{O}(N^3)$, where the other distance measures are calculated in $\mathcal{O}(1)$.

The theoretical advantage of using the shortest paths is enormous: As shown in Figure 8.5, using the shortest paths measure can decrease the coverage time from N to $N/2$. The tree in Figure 8.5a was generated using the shortest paths measure, and the tree in Figure 8.5b was generated using the Manhattan distance measure. Initially, there are twice as many cells below the horizontal corridor compared to the number of cells above this corridor. Note that when generating the subtrees using the shortest paths measure (Figure 8.5a) there are two possible bridges between the trees - one near the initial positions of the robots and one at the endpoint of the subtree. The latter bridge will be chosen with a high probability. In the second tree there is only one possible bridge connecting between the two subtrees. The

distance between the two robots along the first spanning tree path (Figure 8.5a) is $N/2$. The distance between the two robots along the second spanning tree (Figure 8.5b, generated using Manhattan distance) is 8 fine grid cells. Therefore the distance changed from 8 to $N/2$ just by using the shortest paths distance measure.

Yet, in the average case using this measure did not make a difference—the results of running MSTC algorithms on the trees generated using the three distance measures converge. The reason, in our opinion, is that it requires a very specific structure of the terrain in order for the shortest paths measure to make a significant change. One terrain that will gain from using this measure is the one with many corridors (see the example depicted in Figure 8.5). In such a terrain, the difference between the Manhattan distance and the shortest path is significant. Hence we conclude that Create_Tree is adaptable in the sense that the distance measure can be changed in order to fit the terrain.

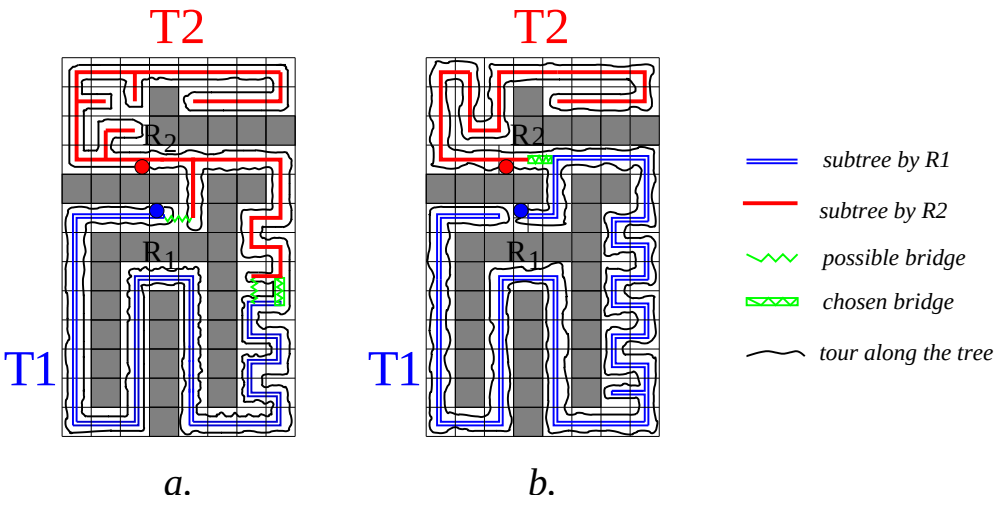


Figure 8.5: An illustration of the trees created using different distance measures by Procedure Create_Tree: a. Manhattan distance and b. Shortest paths.

8.3 Evaluation

We evaluated the effect of the tree construction algorithm Create_Tree on the coverage time obtained by NB_MSTC, B_MSTC and Opt_MSTC. First, we determine the α used by the algorithm. Then we describe extensive simulations of Create_Tree with our chosen α.

8.3.1 Determining α

When connecting k subtrees, the Create_Tree procedure randomly chooses a set of bridges yielding a tree
$\max\{k^\alpha, N\}$ times and the best option between them. We chose the value α empirically to be 2. We noticed that if $\alpha = 2$, then the coverage time obtained by the MSTC family of algorithms decreases substantially. A further improvement can be seen when $\alpha = 3$, but the intensity of the improvement diminishes, more evidently with the results of the Opt_MSTC algorithm (see Figure 8.6).

Note that the time complexity is substantially higher if $\alpha = 3$ and rises from Nk^2 to Nk^3, i.e., an addition of $Nk^2(k-1)$ operations. This becomes critical for large N's and k's.

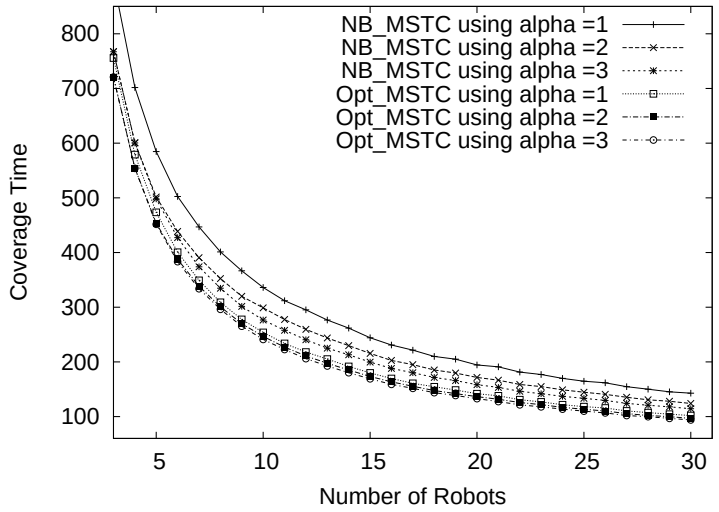


Figure 8.6: Comparing $\alpha = 2$ to $\alpha = 3$ for 1 to 30 robots, when 13% of the area contains obstacles (without disconnecting the area).

8.3.2 Experimental results, $\alpha = 2$

The evaluation of Create_Tree on the coverage time obtained by the family of MSTC algorithms was done while taking two other parameters under consideration. First, the number of robots - from 3 to 30 robots was considered. The second parameter considered was the *density* of obstacles in the terrain, i.e, the ratio between the number of obstacles and the size of the area.

The coverage time obtained by the above algorithms on the trees constructed by all three variants of Create_Tree was compared with the coverage time obtained by the algorithms running on randomly generated spanning trees. The terrain over which the experiment was run was a $20X30$ coarse grid (600 coarse cells, or 2400 fine cells). We first performed the experiment on a grid with no obstacles ("clean" grid), then added at random 40 (6.6%), 80 (13.3%), 100 (16.7%) and 160 (26.7%) obstacles to the coarse grid.

Each trial was run for every number of robots (from 3 to 30) and for every density of obstacles in the terrain. First, we created 300 input lines using each tree construction method: randomly generated trees and Create_Tree generated trees, where each input line represents a random initial distribution of the robots. These input lines were later given to the NB_MSTC, B_MSTC and Opt_MSTC algorithms and the coverage times obtained by these algorithms were compared.

The average coverage times obtained by the algorithms NB_MSTC and Opt_MSTC are presented in Figure 8.7. The results show clearly that the average coverage time obtained by running algorithms NB_MSTC and Opt_MSTC on trees constructed by algorithm Create_Tree are statistically significantly better (using paired two-tailed t-test, the p-value was always less than 10^{-12}) than the average coverage time obtained by the algorithms when run on randomly generated trees. Moreover, the coverage time obtained by running the simplest non-backtracking MSTC algorithm on the trees generated by Create_Tree was, in most cases, even lower than the *optimal* MSTC algorithm run on randomly generated trees. These results were repeated for both dimensions of the experiment: the number of robots and the density of obstacles. The results from running the experiment on B_MSTC are omitted for reasons of clarity of the display, but are compatible with all the other results.

An interesting result was revealed from comparing the improvement in coverage time obtained by the algorithms after performing Create_Tree with different densities of obstacles in the terrain. While the improvement in the

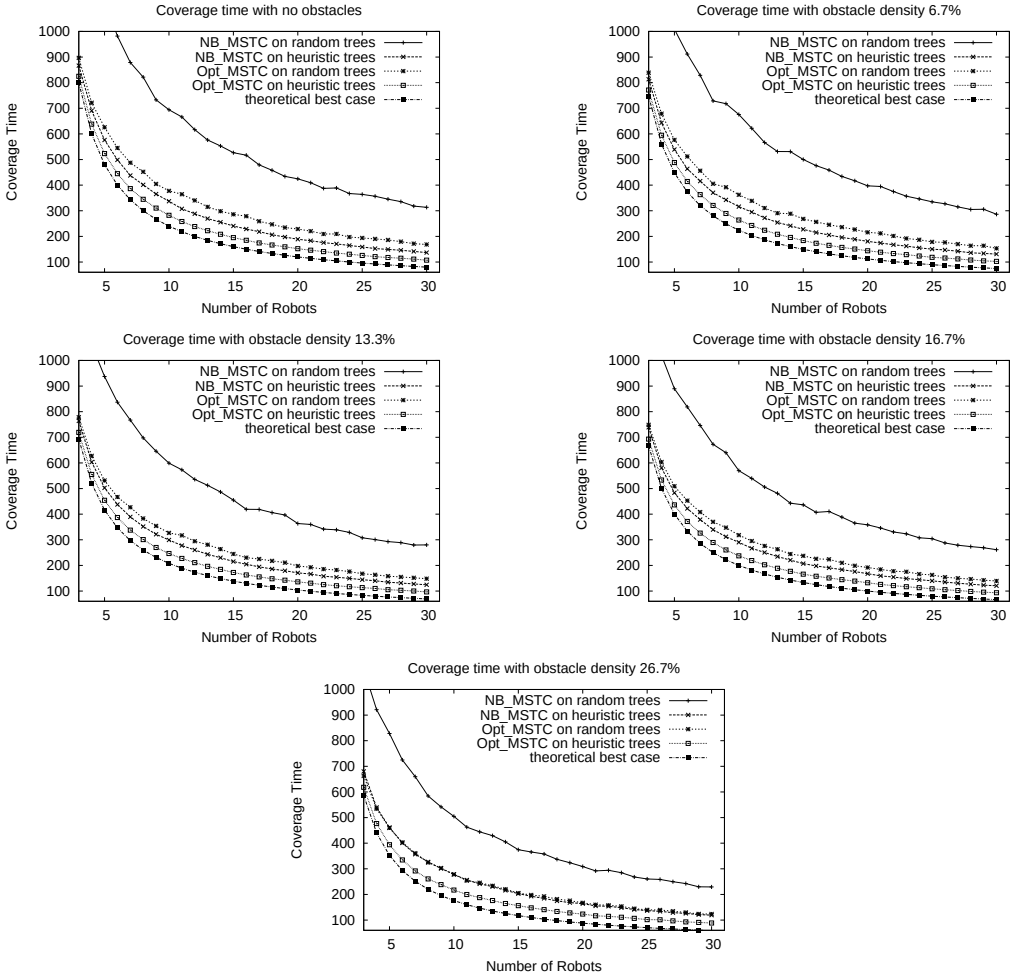


Figure 8.7: Results from comparing coverage time when using random trees vs. trees generated using the Create_Tree algorithm.

coverage time obtained by the algorithms after running Create_Tree remains statistically significant compared to randomly generated trees, as the obstacles become more dense the improvement lessens. For instance, the improvement ratio for 30 robots with no obstacles for the NB_MSTC and Opt_MSTC algorithms are 58% and 38%, respectively. When the density of obstacles is 26% the improvement ratio decreases to 48% and 28% (respectively).

Figure 8.8 presents an example of the improvement ratio in coverage time between Create_Tree generated trees vs. randomly generated trees followed by the execution of the NB_MSTC algorithm. Note that the repetitiveness of the phenomenon is *not* absolute over all number of robots, but the trend is clear. Our initial explanation for the reason of this phenomenon was that the Manhattan distance does not capture the real distance between the robots when more and more obstacles are added to the terrain. However, as stated previously, even when changing the distance measure to the shortest paths the results were not improved. We then deduce that as more and more robots are added and as more obstacles are added to the terrain, there is less freedom in constructing the k spanning trees, i.e., the possibility to spread the subtrees away from each other is limited.

An additional interesting result evolved from a comparison of of the ratio of improvement of the results obtained by the Opt_MSTC and the NB_MSTC algorithms (Figure 8.8). In both cases the improvement ratio from using the Create_Tree generated trees is relatively high, although the NB_MSTC coverage algorithm results in a much higher improvement ratio. This change stems from the fact that if the simple NB_MSTC algorithm is used the change in coverage time is much more evident. The Opt_MSTC algorithm in itself performs with some improvement in coverage time, so there is less to improve from that point.

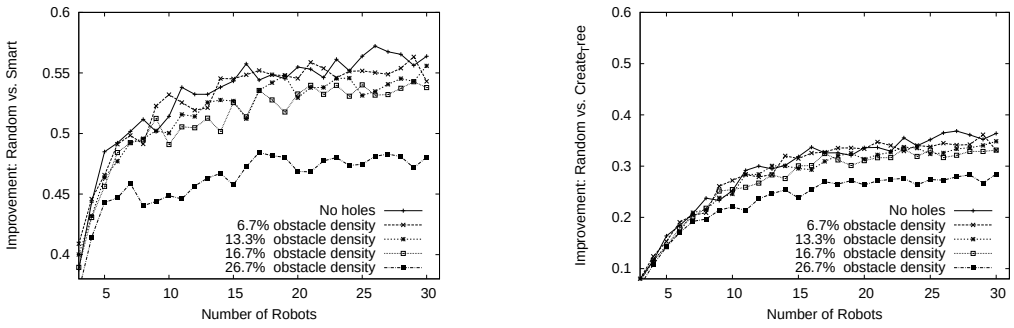


Figure 8.8: Comparison of the improvement ratio in coverage time obtained by algorithm NB_MSTC (left) and Opt_MSTC (right) after generating trees randomly vs. using the Create_Tree algorithm with different densities of obstacles in the terrain.

Chapter 9

Future Directions and Final Remarks

We summarize the key contributions of this thesis in Section 9.1. We discuss future directions for this research in Section 9.2.

9.1 Summary of Key Contributions

In the first and main part of this dissertation we focus on multi-robot patrol in adversarial environments. In this scenario the robots are motivated by the challenge of determining their movement in a way that will maximize their chances of detecting an adversary trying to penetrate through the patrol path. Our contribution in this part of the work is as follows.

- We provide an initial discussion on the problem of multi-robot patrol in adversarial environments, shifting the focus from maximizing frequency criteria (as in previous works) to maximizing the probability of detecting the adversary.
- We provide a polynomial-time algorithm that finds the probability of penetration detection along all points of the patrol, concentrating on perimeter patrol and fence patrol.
- We discuss the implications of the amount of knowledge obtained by the adversary on the patrolling robots on the optimal patrol algorithm chosen by the robots. We describe a polynomial-time algorithm that

finds the optimal patrol for the robots assuming both a full-knowledge adversary and a zero-knowledge adversary.

- We examine the case in which the amount of knowledge the adversary has on the patrol lies somewhere along the knowledge continuum — between full and zero knowledge. We provide both heuristic algorithms for handling this case, and theoretical results that provide the optimal patrol algorithm given the level of uncertainty of the adversarial choice of action.

- We performed an extensive empirical evaluation of the patrol algorithms we developed, where the adversary's role was played by human subjects working against simulated patrolling robots. We examined the different patrol algorithms (heuristic and proven optimal) where the adversary had different amounts of exposed information, representing different adversarial models. The results strengthens the theoretical results, and show that indeed the theoretical optimal patrol algorithms for the zero and full knowledge adversarial environments are optimal also in practice. Moreover, the heuristic algorithms we presented outperformed the algorithm for the full knowledge adversary when only partial knowledge was presented to the player, thus it shows that the common assumption of preparing for the worst-case scenario (here a full knowledge adversary) is *not always* beneficial.

In the second part of the dissertation, we considered two problems in the multi-robot domain. In the first problem, we considered the task reallocation problem in multi-robot formation. In this problem, a team of N robots move in a formation, and k of them need to be extracted from the group. The extraction is done considering the interaction cost between the team members — in our case the cost of sensing inside the formation — where the goal is to minimize the interaction cost between the remaining team members (and thus maximizing the utility function of the remaining group). A summary of our contribution in this chapter is as follows:

- We introduce a new method in which the problem of reallocating k out of N team members to a new task is modeled by a graph, and the utility function is based on the interaction cost between the team members.

- We describe a deterministic algorithm for the reallocation problem which reduces the time complexity of the solution from $\mathcal{O}(N^k)$ to $\mathcal{O}(2^k)$. This result is shown for both cases in which the formation can have either one or more possible leaders.
- We generalize the use of the basic reallocation algorithm for cases in which the utility function has more than one component. In particular, we consider weighted components and prioritized components of the utility function.
- We describe an empirical evaluation of the algorithm and its variations using the Player/Stage simulated environment.
- We show that the method we propose that focuses on the interaction between team members, and the basic algorithm within it, is a general method that can be used in several other domains different from the multi-robot formation domain.

In the last chapter of this dissertation, we consider the problem of improving efficiency of multi-robot coverage. In this problem, a team of robots are required to jointly visit a target area once, and their goal is to minimize the time required to complete the coverage. The paths given to the robots are based on a spanning tree which visits all nodes of the graph that spreads over the target area. The contribution of this part is summarized as follows.

- We initially discuss the impact of the chosen spanning tree on the coverage time of the algorithm.
- We propose a new heuristic algorithm which is based on building efficient spanning trees for coverage.
- We describe an evaluation of this algorithm that demonstrates that when using our heuristic algorithm the resulted coverage time significantly decreases compared to the time required when using random trees (as done in previous work).

9.2 Future Directions

We concentrate on future directions for the first part of this dissertation. Several points have been left open for future work:

The **partial-knowledge adversarial model** should be further examined. In most realistic cases the adversary has neither full knowledge nor zero knowledge on the patrolling robots. We addressed this case in our work by suggesting heuristic algorithms and concentrating on the uncertainty of the adversary's choice of its penetration spot and the reflecting optimal algorithms for the robots. This problem warrants further investigation using other possible tools (for example game theoretic tools), and further empirical evaluation in order to identify other possible proficient approaches to this problem.

Heterogenous robots should be considered. Until know, both in previous work and in our work, the robots have been assumed to be homogenous. New challenges will arise when considering robots that cooperate in the same system but differ in their movement capabilities (movement model or velocity constraints) or sensing capabilities.

Heterogeneous environments from the adversary's point of view should be investigated. In our work we assumed that the penetration time for each segment along the path is identical. Changing penetration times could arise an interesting implications on both the complexity of the calculation (modeling only a small portion of the path does not hold any more) and the algorithm framework (with different weights on different possible actions).

It would be interesting to consider **other sensorial abilities of the robots**. In the extensions of the basic model considered in our work, we addressed only the problem of sensing ahead along the patrol path. An intriguing point to consider is the ability to sense either outside or inside the area, and the impact of characteristics of the environment on this ability (for example in some parts of the area a robot could sense along a certain radius, where in other parts of the area its view might be blocked).

Bibliography

[1] E. U. Acar and H. Choset. Robust sensor-based coverage of unstructured environments. In *International Conference on Intelligent Robots and Systems*, pages 61–68, Maui, Hawaii, USA, 2001.

[2] N. Agmon, N. Hazon, and G. A. Kaminka. Constructing spanning trees for efficient multi-robot coverage. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[3] N. Agmon, G. A. Kaminka, and S. Kraus. Team member-reallocation via tree pruning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, pages 35–40, 2005.

[4] N. Agmon, G. A. Kaminka, and S. Kraus. Multi-robot fence patrol in adversarial domains. In *Proceedings of the Tenth Conference on Intelligent Autonomous Systems (IAS-10)*, pages 193–201. IOS Press, 2008.

[5] N. Agmon, S. Kraus, and G. A. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2008.

[6] N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal of Computing*, 36(1):56–82, 2006.

[7] N. Agmon, V. Sadov, S. Kraus, and G. A. Kaminka. The impact of adversarial knowledge on adversarial planning in perimeter patrol. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08)*, 2008.

[8] M. Ahmadi and P. Stone. A multi-robot system for continuous area sweeping tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[9] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyr. Recent advances on multi-agent patrolling. *Lecture Notes in Computer Science*, 3171:474–483, 2004.

[10] F. Amigoni, N. Gatti, and A. Ippedico. Multiagent technology solutions for planning in ambient intelligence. In *Proceedings of Agent Intelligent Technologies (IAT-08)*, 2008.

[11] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. A distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 15:818–828, 1999.

[12] T. Balch and R. Arkin. Behavior based formation control for mutlirobot systems. *IEEE Transactions on Robotics and Automation*, 14(12):926 – 939, 1998.

[13] M. Batalin and G. Sukhatme. Spreading out, a local approach to multirobot coverage. In *Proceedings of the 6th International Symposium on Distributed Autonomous Robotic Systems*, pages 373–382, 2002.

[14] E. Brunner, S. Domhof, and F. Langer. Nonparametric analysis of longitudinal data in factorial experiments. *Journal of Optimization Theory and Application staff,*, 114(1):243–244, 2002.

[15] Z. Butler, A. Rizzi, and R. L. Hollis. Complete distributed coverage of rectilinear environments. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, March 2000.

[16] Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proceedings of Agent Intelligent Technologies (IAT-04)*, 2004.

[17] P. Chinn, R. Grimaldi, and S. Heubach. The frequency of summands of a particular size in palindromic compositions. *Ars Comb.*, 69, 2003.

[18] H. Choset. Coverage for robotics - a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113–126, 2001.

[19] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving the robots gathering problem. *Lecture Notes in Computer Science*, 2719:1181 – 1196, 2003.

[20] J. Colegrave and A. Branch. A case study of autonomous household vacuum cleaner. In *AIAA/NASA CIRFFSS*, 1994.

[21] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir. Random walks on weighted graphs and applications to on-line algorithms. *Journal of the ACM*, 40(3), 1993.

[22] T. Corman, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[23] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[24] V. D. Dang and N. R. Jenings. Generating coalition structures with finite bound from the optimal guarantees. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 564–571, 2004.

[25] J. P. Desai. A graph theoretic approach for modeling mobile robot team formation. *Journal of Robotic Systems*, 19(11):511–525, 2001.

[26] J. L. Devore. *Probability and Statistics for Engineering and the Sciences*. Brooks/Cole Publishing Company, 1991.

[27] M. B. Dias and A. Stentz. A free market architecture for distributed control of a multirobot system. In *Proceedings of the Sixth Conference on Intelligent Autonomous Systems (IAS-6)*, pages 115–122, 2000.

[28] M. B. Dias, R. Zlot, M. Zinck, J. P. Gonzalez, and A. Stentz. A versatile implementation of the traderbots approach for multirobot coordination. In *Proceedings of the Eighth Conference on Intelligent Autonomous Systems (IAS-8)*, 2004.

[29] M. F. Duarte and Y. H. Hu. Distance-based decision fusion in a distributed wireless sensor network. *Telecommunication Systems*, 26(2-4):339–350, 2004.

[30] Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007.

[31] Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Math and Artificial Intelligence, to appear*, 2009.

[32] Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Math and Artificial Intelligence journal (AMAI)*, 2009, to Appear.

[33] Y. Elmaliach, A. Shiloni, and G. A. Kaminka. A realistic model of frequency-based multi-robot fence patrolling. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08)*, 2008.

[34] J. Feddema, C. Lewis, and D. Schoenwald. Decentralized control of cooperative robotic vehicles: theory and application. *IEEE Transactions on Robotics and Automation*, 18:852– 864, 2002.

[35] E. Ferranti, N. Trigoni, and M. Levene. Brick and mortar - an on-line multi-agent exploration algorithm. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007.

[36] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Sciense*, 407(1-3):412–447, 2008.

[37] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8(3):325–344, 2000.

[38] J. Fredslund and M. Mataric. A general algorithm for robot formation using local sensing and minimal communication. *IEEE Transactions on Robotics and Automation*, 18:837–846, 2002.

[39] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):77–98, 2001.

[40] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry*, 24:197–224, 2003.

[41] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[42] B. P. Gerkey and M. J. Matarić. Murdoch: publish/subscribe task allocation for heterogeneous agents. In *Proceedings of the fourth international conference on Autonomous agents (AGENTS-00)*, pages 203–204, 2000.

[43] B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23:939–954, 2004.

[44] B. P. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project - tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, pages 317–323, Coimbra, Portugal, Jul 2003.

[45] T. Haynes and S. Sen. Evolving behavioral strategies predators and prey. In *IJCAI-95 Workshop on Adaptation and Learning in Multiagent Systems*, pages 32–37, 1995.

[46] N. Hazon and G. A. Kaminka. Redundancy, efficiency and robustness in multi-robot coverage. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[47] N. Hazon and G. A. Kaminka. On redundancy, efficiency, and robustness in coverage for multiple robots. *Robotics and Autonomous Systems, to appear*, 2008.

[48] S. Hedberg. Robots cleaning up hazardous waste. *AI Expert*, pages 20–24, May 1995.

[49] Y. Huang, Z. Cao, and E. Hall. Region filling operations for mobile robot using computer graphics. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA-86)*, pages 1607–1614, 1986.

[50] L. Iocchi, D. Nardi, and M. Salerno. Reactivity and deliberation: A survey on multi-robot systems. In *Balancing Reactivity and Social Deliberation in Multi-Agent Systems, From RoboCup to Real-World Applications (selected papers from the ECAI 2000 Workshop and additional contributions)*, pages 9–34, 2001.

[51] G. A. Kaminka, R. Schechter-Glick, and V. Sadov. Using sensor morphology for multirobot formations. *IEEE Transactions on Robotics*, 24(2):271–282, 2008.

[52] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. Robocup: A challenge problem for ai and robotics. In *RoboCup-97: Robot Soccer World Cup I*, pages 1–19, 1998.

[53] C. S. Kong, A. P. New, and I. Rekleitis. Distributed coverage with multi-robot system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[54] S. Kraus, J. S. Rosenschein, and M. Fenster. Exploiting focal points among alternative solutions: Two approaches. *Annals of Math and Artificial Intelligence*, 28(1–4):187–258, 2000.

[55] S. Kumar and P. R. Cohen. Towards a fault-tolerant multi-agent system architecture. In *Proceedings of the fourth international conference on Autonomous agents (AGENTS-00)*, pages 459–466, 2000.

[56] M. Lemay, F. Michaud, D. Letourneau, and J. M. Valin. Autonomous initialization of robot formation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[57] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 1947.

[58] N., N. Hazon, and G. A. Kaminka. The giving tree: Constructing trees for efficient offline and online multi-robot coverage. *Special issue of the Annals of Math and Artificial Intelligence journal (AMAI) on Multi-Robot Coverage, Search, and Exploration*, 2009, to appear.

[59] D. J. Naffin and G. S. Sukhatme. Negotiated formations. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, pages 181–190, Mar 2004.

[60] E. Osherovich, V. Yanovski, W. I. A, and A. M. Bruckstein. Robust and efficient covering of unknown continuous domains with simple, ant-like a(ge)nts. Technical report, Technion, Israel, 2007.

[61] L. E. Parker. Current research in multirobot systems. *Artificial Life and Robotics*, 7, 2003.

[62] P. Paruchuri, J. P. Pearce, M. Tambe, F. Ordonez, and S. Kraus. An efficient heuristic approach for security against multiple adversaries. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08)*, 2007.

[63] P. Paruchuri, M. Tambe, F. Ordonez, and S. Kraus. Security in multiagent systems by policy randomization. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-07)*, 2007.

[64] J. Pita, M. Jain, F. Ordonez, , M. Tambe, S. Kraus, and R. Magorii-Cohen. Effective solutions for real-world stackelberg games: When agents must deal with human uncertainties. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, 2009.

[65] M. A. Rahman, M. S. Miah, W. Gueaieb, and A. E. Saddik. Senora: A p2p service oriented framework for collaborative multi-robot sensor network. *IEEE Sensors Journal, Special Issue on Intelligent Sensors*, 7(5):658–666, 2007.

[66] I. Rekleitis, G. Dudek, and E. Milios. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *International Joint Conference in Artificial Intelligence (IJCAI-97)*, volume 2, pages 1340–1345, Nagoya, Japan, August 1997. Morgan Kaufmann Publishers, Inc.

[67] I. Rekleitis, G. Dudek, and E. Milios. Multi-robot collaboration for robust exploration. *Annals of Mathematics and Artificial Intelligence*, 31:7–40, 2001.

[68] I. Rekleitis, V. Lee-Shue, A. P. New, and H. Choset. Limited communication, multi-robot team based coverage. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3462–3468, 2004.

[69] T. Sak, J. Wainer, and S. K. Goldenstein. Probabilistic multiagent patrolling. In *Proc. of the 19th Brazilian Symposium on Artificial Intelligence (SBIA-08)*, pages 124–133, 2008.

[70] P. V. Sander, D. Peleshchuk, and B. J. Grosz. A scalable, distributed algorithm for efficient task allocation. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 1191–1198, 2002.

[71] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.

[72] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, pages 209–238, 1999.

[73] T. Sandholm and V. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94(1):99–137, 1997.

[74] S. Sen and S. Dutta. Searching for optimal coalition structures. In *Proceedings of the Fourth International Conference on Multiagent Systems*, pages 287 – 292, 2000.

[75] K. C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. In *Proceedings of ACM Conference on Measurement and Modeling of Computation Systems*, pages 171–180, May 1989.

[76] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.

[77] J. S. Shieh and T. W. Calvert. View and route planning for patrol and exploring robots. *Advanced Robotics*, 6(4):399–430, 1992.

[78] A. Shiloni, N. Agmon, and G. A. Kaminka. Of robot ants and elephants. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, 2009.

[79] T. C. H. Sit, Z. Liu, M. H. A. Jr., and W. K. G. Seah. Multi-robot mobility enhanced hop-count based localization in ad hoc networks. *Robotics and Autonomous Systems*, 55(3):244–252, 2007.

[80] S. Stramigioli. Special issue on mobile multirobot systems. *IEEE Robotics and Automation Magazine*, 15(1), 2008.

[81] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[82] J. Svennebring and S. Koenig. Building terrain-covering ant robots - a feasibility study. *Autonomous Robots*, 16(3):313–332, 2004.

[83] M. Tambe and W. Zhang. Towards flexible teamwork in persistent teams: extended report. *Journal of Autonomous Agents and Multi-Agent Systems*, 3:159–183, 1998.

[84] P. Tosic and G. Agha. Maximal clique based distributed group formation for autonomous agent coalitions. In *Coalitions and Teams Workshop, AAMAS*, 2004.

[85] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry. Probabilistic pursuit-evasion games - theory, implementation, and experimental evaluation. *Robotics and Automation, IEEE Transactions on*, 18(5):662–669, 2002.

[86] I. Wagner, M. Lindenbaum, and A. Bruckstein. Mac vs. pc determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *International Journal of Robotics Research*, 19(1):12–31, 2000.

[87] I. A. Wagner and A. M. Bruckstein. From ants to a(ge)nts: A special issue on ant-robotics. *Annals of Math and Artificial Intelligence*, 31(1-4):1–5, 2001.

[88] X. Zheng, S. Jain, S. Koenig, and D. Kempe. Multi-robot forest coverage. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.