Ben-Gurion University of the Negev
The Faculty of Engineering Sciences
The Department of **Industrial Engineering And Management**

Distributed Optimization Challenges in Mobile Sensor Team Applications

Arseni Pertzovskiy

Thesis submitted in partial fulfillment of the requirements
for the Master of Sciences degree

Under the supervision of **Prof. Roie Zivan** and **Prof. Dan Hermelin**

February 2021

אוניברסיטת בן־גוריון בנגב
הפקולטה למדעי ההנדסה
המחלקה למהבנדסת תעשיה וניהול

אתגרי אופטימיזציה מבוזרת באפליקציות הכוללות קבוצת סנסורים ניידים

ארסני פרצובסקי

חיבור לשם קבלת התואר "מגיסטר" בפקולטה למדעי הטבע

בהנחיית מנחה פרופ' רועי זיוון ופרופ' דן הרמלין

פברואר 2021

Ben-Gurion University of the Negev
The Faculty of Engineering Sciences
The Department of **Industrial Engineering And Management**

Distributed Optimization Challenges in Multi-Agent Applications

Arseni Pertzovskiy

Thesis submitted in partial fulfillment of the requirements
for the Master of Sciences degree

Under the supervision of **Prof. Roie Zivan** and **Prof. Dan Hermelin**

Signature of student: _____ Date: _____

Signature of supervisor: _____ Date: _____

Signature of chairperson of the

committee for graduate studies: _____ Date: _____

February 2021

Abstract

Multi-agent applications often require agents to take actions to achieve a common goal. Coordinating a mobile sensor team (MST) to cover targets exemplifies a challenging multi-agent optimization problem. The distributed constraint optimization problem (DCOP) is a general framework for describing distributed problems that include constraints, which can be represented by a graphical model and solved by using message-passing algorithms. In many realistic multi-agent applications, such as the MST coverage problem, various properties of the problem are not compatible with the abstract DCOP model. For DCOP to apply to realistic scenarios, the model must be extended and the algorithms designed to solve the extended model. For example, a variation of the DCOP model was recently adjusted to represent problems including MSTs (DCOP_MSTs), and incomplete algorithms such as Max-sum were enhanced with exploration methods to solve these problems. In DCOP_MST, agents maintain variables for their physical positions, whereas each target is represented by a constraint that reflects the quality of coverage of the given target. Unfortunately, the proposed algorithms do not prevent collisions between mobile sensors. In this first attempt to extend DCOP models and algorithms to include realistic features, we propose a collision-avoiding version of the

Max-sum algorithm (CAMS), in which function nodes representing hard constraints are added to the factor graph generated in each iteration to prevent the selection of a single location by more than one agent. For small scenarios, we prove that the proposed algorithm converges to the optimal solution and present empirical evidence that, in more complex scenarios, the proposed algorithm maintains high-quality coverage while avoiding collisions. Our empirical study includes both software simulations and experiments involving a team of sensor-carrying robots. Finally, we outline additional realistic properties of such applications that we intend to address in future research.

Keywords: Multi-agent optimization, Real-world applications, Incomplete distributed algorithms

Acknowledgments

I am indebted to Professor Roie Zivan and Professor Dan Hermelin, who have helped me and taught me a lot. Thank you for the dedicated guidance, patience, and encouragement. I would like to thank my family for all support they gave me. A special thanks goes to the faculty and to the university, who gave me an opportunity to do this research.

Contents

List of Figures

List of Tables

1 Introduction

The development of abstract distributed models and algorithms leaves us now with the task of representing and solving them in real-world situations. Unfortunately, in numerous real-world cases, these abstract models and algorithms are not applicable due to inconsistencies with aspects of the application. Thus, extensions to these models and algorithms must be developed to adapt them to such real-world scenarios.

Some of the most challenging multi-agent systems involve teams of mobile sensing agents whose job is to acquire information in a given area. Examples include networks of sensors [9, 44] and rescue teams in disaster areas [13]. A crucial, common feature of these applications is that agents select physical locations to move to, and this selection affects their future interactions. For example, if a mobile sensor decides to sense a given area, it will coordinate its actions with nearby sensors.

Such scenarios have been previously modeled using the framework of the distributed constraint optimization problem (DCOP) in which mobile sensors are represented as agents and their tasks or targets as constraints [29]. However, if all possible future moves of dynamic agents are considered, the problem becomes dense. To deal with this difficulty, previous work has suggested an iterative process in which a DCOP instance is built into

each iteration to represent the current situation (e.g., sensor positions) and in which only limited agent movements are considered. Agents run a distributed algorithm (which might involve several communication cycles) to select the best next joint move and, after executing it, they build a new DCOP instance that takes into account their new position [29].

Zivan et al. proposed an extension of the DCOP model and the corresponding local search algorithms for representing and solving such scenarios, particularly focusing on teams of mobile sensing agents that need to select a deployment for the sensors to cover a partially unknown environment. This is denoted *DCOP_MST*. DCOP_MST allows agents to adjust their location to adapt to dynamically changing environments [44].

The Max-sum algorithm [1, 3, 7] has been the subject of intensive study for solving the DCOP and has been applied to many realistic applications, including sensor networks [6], smart homes [25], and teams of rescue agents [23]. In contrast with standard local-search algorithms, agents in Max-sum do not propagate assignments but instead calculate utilities (or costs) for each variable, considering all possible values of the variables of their neighboring agents. The general structure of the algorithm is exploitative; in other words, the agents attempt to compute the best costs or utilities for possible value assignments according to their own problem data and recent information received via messages from their neighbors. A version of Max-sum was also proposed for solving DCOP_MST [44]. The main challenge in applying Max-sum to such problems is to overcome the inherent symmetry generated by the algorithm, which is done either by attracting all neighboring sensors to cover a target, resulting in inefficient use of sensors, or by encouraging all neighboring sensors to explore the area instead of covering the target. Reference [35] proposed methods for

breaking this symmetry and finding an equilibrium between exploration and exploitation.

In this work, we address a different limitation on teams of mobile sensing agents residing on hardware robots, namely, the need to avoid collisions. This requirement has been ignored by all previous studies of DCOP_MST. To allow mobile sensors to explore an area, search for targets, and select a deployment that maximizes team coverage without having robots collide, we propose the collision-avoidance Max-sum (CAMS) algorithm. As in the standard Max-sum algorithm, the problem in CAMS is represented by a factor graph, which is a bipartite graph containing nodes that represent variables and functions (i.e., constraints) such that nodes from one type only have neighbors of the other type. As in previous attempts to use Max-sum to solve problems including dynamic agents, in every iteration (before each movement of the agents), a factor graph is constructed, the algorithm is executed for a limited number of steps,[1] and then the agents select the next location to which to move. The novelty in the present work is the new type of function-nodes added to these factor graphs and that represent locations to which agents can move, which contrasts with factor graphs generated by agents in Max-sum_MST that only include function-nodes representing targets [35]. A function node representing a location to which more than one agent can move excludes this option by assigning it a utility of $-\infty$.

This work is structured as follows: Chapter 2 describes the background; Chapter 2.8 goes through the preparation and implementation; Chapter 3

[1] To avoid confusion, we use the term *iteration* for each phase in Max-sum_MST in which agents select their next location, and *steps* for iterations of the Max-sum algorithm used to solve the factor graph generated in the given phase.

describes the CAMS algorithm and presents the results; Chapter 4 summarizes this work; and, finally, Chapter 5 discusses the next stages of this research.

2 Previous Work & Background

This chapter presents the DCOP, the extended DCOP_MST model, and distributed incomplete algorithms and their adjustments for the DCOP_MST model. In addition, the robot operating system (ROS) and Hamster robot hardware are introduced.

2.1 The Distributed Constraint Optimization Problem

Distributed constraint optimization is a general formulation of multi-agent coordination problems that have previously been used for static sensor networks and many other applications. A distributed constraint optimization problem (DCOP) is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ is a finite set of agents, $\mathcal{X} = \{X_1, X_2, \ldots, X_m\}$ is a finite set of variables, $\mathcal{D} = \{D_1, D_2, \ldots, D_m\}$ is the set of finite domains for the variables, and $\mathcal{C}$ is a finite set of constraints.[1] Each variable X_i is controlled (or owned) by an agent who assigns it a value from the finite set of values D_i; each

[1] Constraints are typically partitioned into hard constraints represented by relations and soft constraints represented by cost functions. Here, we do not consider hard constraints and use only cost functions.

agent may control multiple variables. Each constraint $C \in \mathcal{C}$ is a function $C : D_{i_1} \times D_{i_2} \times \cdots \times D_{i_k} \rightarrow \mathbb{R}_+ \cup \{0\}$ that maps assignments of a subset of the variables (called the scope of the constraint) to a non-negative *cost*. The cost of a *complete assignment* of values to all variables is computed by summing the costs of all constraints. A solution of a DCOP is a complete assignment (i.e., each variable in X is assigned a value). The optimal solution is the solution with minimum cost (or with maximal utility in the case of a maximization problem).

Control in DCOPs is distributed, with agents only able to assign values to variables that they possess. Furthermore, agents have knowledge only of the constraints involving their own variables. Coordination is achieved by passing messages. A standard assumption is that agents exchange messages only with a subset of the other agents, called their *neighbors*. Agent A_i and agent A_j are neighbors if and only if at least one constraint exists whose scope includes a variable controlled by A_i and a variable controlled by A_j. While message transmission may be delayed, it is assumed that messages sent from one agent to another are received in the order that they were sent [14] [42].

2.2 DCOP for Mobile Sensing Teams

The DCOP model makes several assumptions that do not hold in MST applications. For example, it assumes that the neighbor set of agents is constant. It also assumes that the constraints (i.e., the utilities or costs incurred by each partial assignment) are known *a priori* and are constant. Mobile sensors, in contrast, are dynamic by nature. The movement of agents constantly changes the neighbor set. A realistic environment un-

dergoing changes modifies the set of constraints and, consequently, modifies the utilities for deployment decisions. Consequently, dynamic elements must be formalized and integrated into the DCOP model for it to apply to MSTs.

The DCOP for MSTs (DCOP_MST) is a dynamic DCOP formulation that models the MST coordination problem [44]. The agents $A = \{A_1, A_2, \ldots, A_n\}$ in a MST are physically situated in the environment and are modeled as a metric space with distance function d. The *current position* of agent A_i is denoted cp_i; we assume that this position is accurately known by the agent. Locations (or positions) that can be occupied by agents constitute a finite set of discrete points that form a subset of the total environment. These points can either be (i) a discretization of the underlying space or (ii) locations that dominate other nearby points in terms of the sensing quality they afford agents located there. In Figure 2.1, the environment is a Euclidean plane, agents are depicted as small robots, and possible locations are represented by X. Time is discretized so that agents can compute movements between possible positions. The maximum distance that A_i can travel in a single time step is its *mobility range* mr_i. The mobility range of each agent is represented in Figure 2.1 by the dashed, outer circle centered on each agent. All X within the circle are locations to which the agent can move from its current position in a single time step.

Agents can effectively sense targets only within a limited *sensing range* sr_i. Given the sensing-range constraint, each agent A_i can observe all targets within a distance sr_i from cp_i but cannot observe any more distant target. The sensing range is depicted in Figure 2.1 by the darker, inner circle centered on each agent.

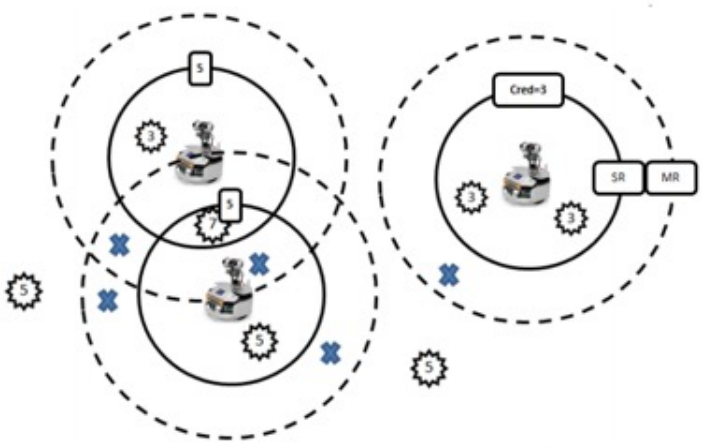


Figure 2.1: Example with three agents. Dashed outer rings around each agent depict the agent's mobility range. Dark inner rings show the sensing range with the numeric credibility of the agent. Stars represent the targets with their coverage requirement. X depicts possible locations where agents may position themselves.

Agents may also differ in the quality of their sensing abilities, which is called their "credibility." The credibility of agent A_i is denoted by the positive real number $cred(i)$, with greater values indicating better sensing ability. $cred(i)$ is exogenously provided (for instance, calculated by a reputation model) and accurately represents the agent's sensing ability. In Figure 2.1, the credibility of each agent is presented as a number in a square on the agent's sensing-range circle [33].

The credibility of individual agents sensing the same target are combined by using a joint credibility function $F : 2^A \rightarrow \mathbb{R}$, where 2^A denotes the power set of A. F must be monotonic so that additional sensing agents can only improve the joint credibility. Formally, for two sets $S' \subseteq S \subseteq A$, we require that $F(S) \leq F(S_0)$.

Targets $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$ are represented implicitly by the *environmental requirement* function ER, which maps each point in the environment to a non-negative real number representing the degree of coverage (which we define shortly) required for that point to be adequately sensed. In this representation, targets are the points p with $ER(p) > 0$. Because targets may arise, move, or disappear, ER changes dynamically. Moreover, ER can change as the agent team becomes aware of new targets. A major aspect of the MST problem is to explore the environment sufficiently to be aware of targets. In the example presented in Figure 2.1, seven targets are shown as serrated circles and their numbers represent their ER values. Agents within the sensing range of a target p are said to cover the target. Given a target p, the set of agents within sensing range of p is

$$sr_p = \{A_i \in A | d(p, cp_i) \leq sr_i\}.$$

The remaining coverage requirement of target p is the environmental requirement of p diminished by the joint credibility of the covering agents, down to a minimum value of zero:

$$cur\_req(p) = max\{0, ER(p) \ominus F(sr_p)\},$$

where $\ominus : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is a binary operator (written in infix notation) that decreases the environmental requirement by the joint credibility. For $x, y, z \in \mathbb{R}$ with $y > z$, we require that $x \ominus y < x \ominus z$, so that decreasing the environmental requirement by a higher joint credibility lowers the remaining coverage requirement. In this work, $\ominus$ will be the standard subtraction operator [43].

The constraint C_T for a target T only involves those agents A_i whose domain of variables includes a location within the sensing range sr_i of T.

Therefore, as the domains change, the constraints change as well, so the set of neighbors for each agent changes over time as the agents move. In DCOP_MST, two agents are neighbors if their sensing areas overlap after the agents both move toward each other the greatest distance possible in a single time step. This encodes the fact that two such agents might directly influence each other (e.g., by observing the same target in the next time step).

The *local environment* of agent A_i is the joint area within sr_i from all positions within mr_i from cp_i. Specifically, denoting the set of neighbors of A_i by $curr\_nei_i$, we formalize this by

$$cur\_nei_i = \{A_j | d(cp_i, cp_j) \leq mr_i + mr_j + sr_i + sr_j\}.$$

Because agents can only communicate with their neighbors, agents in DCOP_MST can only communicate with other agents who are physically nearby.

The global goal of agents is to position themselves to minimize

$$F_{sum}(\mathcal{T}) = \sum_{T_i \in \mathcal{T}} cur\_req(T_j).$$

In some cases, the values of *cur_req* may be reduced to zero for all targets, indicating perfect coverage. However, in other cases, this may not be possible (e.g., because of insufficient numbers or quality of agents). In these cases, we strive to minimize the sum of the remaining coverage requirements for all targets. Such a minimization problem is NP-hard [31]. Another possible objective would be to minimize the maximum remaining coverage requirement over all targets. Note that the model and the techniques presented here could be applied to achieve this objective; however, to simplify the presentation, we do not discuss this here.

2.3 Standard Max-sum

Max-sum [7] operates on a *factor graph*, which is a bipartite graph that includes nodes representing variables and constraints [10]. Each variable-node representing a DCOP variable is connected to all function-nodes that represent constraints in which the node is involved. Variable-nodes and function-nodes are considered "agents" in Max-sum; that is, they can send and receive messages and compute.

A message sent to or from variable-node X (for simplicity, we use the same notation for the variable and the variable-node representing it) is a vector of size $|D_X|$ (the size of X's domain, D_X) including a cost (belief) for each value in D_X. Before the first iteration, all nodes assume that all messages they previously received (in iteration 0) include vectors of zeros. A message sent from variable-node X to function-node F in iteration $i \geq 1$ is formalized as follows:

$$Q^i_{X \to F} = \sum_{F' \in F_X, F' \neq F} R^{i-1}_{F' \to X} - \alpha,$$

where $Q^i_{X \to F}$ is the message that variable-node X intends to send to function-node F in iteration i, F_X is the set of function-node neighbors of variable-node X, and $R^{i-1}_{F' \to X}$ is the message sent to variable-node X by function-node F' in iteration $i-1$. α is a constant that is reduced from all costs included in the message (i.e., the beliefs intended for each $x \in D_X$) to prevent the costs carried by messages throughout the algorithm run from growing arbitrarily large.

A message $R^i_{F \to X}$ sent from a function-node F to variable-node X in iteration i includes for each value $x \in D_X$ $min_{PA_{-X}} cost(\langle X, x \rangle, PA_{-X})$, where PA_{-X} is a possible combination of value assignments to variables involved

in F, not including X. The term $cost(\langle X, x\rangle, PA_{-X})$ represents the cost of a partial assignment $a = \{\langle X, x\rangle, PA_{-X}\}$, which is $f(a) + \sum_{X' \in X_F, X' \neq X, \langle X', x'\rangle \in a} (Q^{i-1}_{X' \to F})_{x'}$, where $f(a)$ is the original cost in the constraint represented by F for the partial assignment a, X_F is the set of variable-node neighbors of F, and $(Q^{i-1}_{X' \to F})_{x'}$ is the cost that was received in the message sent from variable-node X' in iteration $i-1$ for the value x' that is assigned to X' in a. X selects its value assignment $\hat{x} \in D_X$ after iteration k as follows:

$$\hat{x} = \arg\min_{x \in D_X} \sum_{F \in F_X} (R^k_{F \to X})_x.$$

Assuming that no tied beliefs exist, Max-sum converges in linear time to the optimal solution when solving problems represented by a tree-structured factor graph [17].[2] When Max-sum operates on a single-cycle factor graph, it will reach a state in which it repeatedly follows a maximal (or minimal for minimization problems) path of assignments in the cycle. The algorithm converges to the optimal solution if and only if this path is consistent (i.e., includes a single value assignment for each variable [8]). For a more detailed description of the standard Max-sum algorithm, please see Ref. [41]. The Max-sum algorithm has been the subject of intense study in DCOPs and has been applied to numerous realistic applications, including mobile sensor networks [4, 5, 19, 24, 28–30] and teams of rescue agents [13, 23].

[2]Ties can be avoided by adding for each variable-node a unary constraint with extremely small random utilities [7].

2.4 Adjusting Max-sum Algorithm to DCOP_MST

Although Ref. [44] proposed the Max-sum algorithm for solving DCOP_MST, it was only in its basic form and made no effort to explore methods to improve the runtime. In contrast with standard local search algorithms, agents in Max-sum do not propagate assignments but instead calculate utilities (or costs) for each variable, taking into account all possible value assignments for the variables of their neighboring agents. Since the computation performed by Max-sum is exponential in the number of agents involved in a constraint, constraints that involve many agents (k-ary) represent a computational bottleneck. Thus, increasing the number of agents that can be assigned to a task would prevent the Max-sum algorithm from solving such problems. Clearly, extending the local environment of agents results in agents being effective for more tasks or targets and in a larger constraint arity. Therefore, in dynamic scenarios with limited time to reach decisions, Max-sum is effective either for problems that do not require exponential computations by targets or for problems with limited constraint arity (i.e., where the local environment of agents is limited). Yedidsion et al. proposed a novel exploration method specifically designed for Max-sum and based on meta-reasoning: for each target, agents select a subset of the sensors that can be effective for covering it [36]. The proposed function meta-reasoning (FMR, discussed later in this section) method breaks the relation between the size of the local environment of agents and the arity of the constraints. In other words, the arity of the constraint is not defined by the number of sensors within the sensing range of target t after the next assignment selection (i.e., the "neighbors" of t), but rather by the number of sensors required for covering t. Although assignment selections are not

a part of an original Max-sum algorithm, assignment selections determine the local environments in DCOP_MST and directly affect the structure of the constraint network (and, consequentially, the factor graph).

Agents in Max-sum_MST perform as follows:

1. Select a random assignment.
2. Generate a factor graph according to the current assignment where each sensor is a variable-node and each target is a function-node. Variable-node i is connected by an edge to a function-node if and only if the distance between them is less than or equal to the sum $mr_i + sr_i$ (i.e., the sensor can cover the target after a single move).
3. The agents execute the Max-sum algorithm for a predefined number of iterations.
4. The sensors move to the best position (value assignment) as calculated by the algorithm.
5. A new factor graph is generated according to the new assignment selection, and then the process repeats itself.

We now discuss how we solve the challenges that arise upon applying Max-sum_MST to the DCOP_MST model.

2.4.1 Handling Runtime

In our case, agents select the locations from which they derive the highest utility (i.e., from which they are most effective). The next factor graph is generated considering the new locations of the agents.

The number of message cycles executed before an assignment (position) selection must be chosen with care. On the one hand, we want the information regarding the coverage capabilities of sensors to propagate to other sensors. On the other hand, these Max-sum message cycles result in a single movement for the sensors, so we want to avoid unnecessary delays. The results of Yedidsion's experiments show that a small number of message cycles (five iterations) suffices to maximize performance. Our results indicate that more cycles are needed to get good performance. We used 30 iterations in our experimental setup.

Regarding the messages of the function-nodes, the only information required by the function to compute the utility is whether a sensor covers the target (i.e., calculations can be executed for only two types of positions). We can thus apply here the method known as "fast Max-sum" [29], which reduces the complexity for a function-node to generate a message to $O(D * 2^{K-1})$.

2.4.2 Handling Exploration: Function Meta Reasoning

The FMR method tackles a common property of DCOP_MST whereby targets have more neighbors than required for covering them. Consider an iteration i in which the factor graph FG_i is generated based on the locations of sensors selected in iteration $i-1$. Denote by $n(t)_i$ the set of neighboring sensors of target t in FG_i and by $cred_{n(t)_i}$ the total credibility of $n(t)_i$. Furthermore, denote by $r(t)_i$ a subset of $n(t)_i$ and by $cred_{r(t)_i}$ the total credibility of $r(t)_i$. When there exists a subset $r(t)_i$ for which the importance of target t is less than $cred_{r(t)_i}$, t can select $r(t)_i$ neighbors to cover it and allow the other $n(t)_i - r(t)_i$ neighbors to explore. We implement this by

generating a new factor graph $\widehat{FG_i}$ in which each target t has at most $r(t)_i$ neighbors. This can be done distributively by having each target t remove the edges between it and $n(t)_i - r(t)_i$ of its neighbors. For homogeneous agents and targets, where $r(t)$ is the required number of sensors to cover target t, $r(t)$ is constant.

Yedidsion et al. proposed the following greedy heuristic for function-node t to select $|r(t)|$ neighbors for which $|n(t)_i| > |r(t)|$ [33]. The heuristic is tuned as follows with respect to the type of joint credibility function used:

1. Each of the $n(t)_i$ sensor neighbors sends to t its degree in FG_i (i.e., the number of function-node neighbors it has in FG_i).

2. t divides its $n(t)_i$ neighbors into two subsets: $\hat{n}(t)_i$ and $\bar{n}(t)_i$. $\hat{n}(t)_i$ includes all neighbors that are currently within sensing range of t and $\bar{n}(t)_i$ includes the remaining neighbors.

3. While $|n(t)_i| > |r(t)|$:

   (1) If $\bar{n}(t)_i \neq \emptyset$, remove from $n(t)$ the neighbor in $\bar{n}(t)_i$ with the highest degree.

   (2) Else, remove from $n(t)$ the neighbor in $\hat{n}(t)_i$ with the lowest degree.

Note that, when using this method, the complexity for producing each of the messages to be sent by the function-node to its neighbors is no longer exponential in $|n(t)_i| - 1$ as in standard Max-sum; instead, it is exponential in $|r(t)_i| - 1$. Thus, the complexity of computing function-nodes no longer depends on the sensing and mobility ranges of the sensors.

2.4.3 Handling Tie Breaking

The FMR method breaks the connections between targets and agents in a position that allows them to sense the target (although there is no need for the agents to do so). The objective of this rupture is to encourage these agents to search for other targets where their sensing is required. However, the agents may equally well stay in their current location or select a new location.

Thus, to stimulate exploration, agents select assignments randomly from among the values tried that offer the highest utility. This method is denoted *Rand* and allows agents to continuously explore new positions and the targets that may be covered from these positions. In standard Max-sum (solving standard DCOPs), the objective of tie-breaking is simply to ensure that agents select the same solution. In DCOP_MST, on the other hand, agents are incentivized to seek locations from which they are more effective via the tie-breaking method. Although it enhances exploration, the *Rand* method has a dichotomous effect in terms of the overall performance of the team.

In the following chapters we denote Max-sum_FMR-*Rand* as *Max-sum_MST*.

2.5 Robot Operating System

The Robot Operating System (ROS) is an open-source framework designed to serve as a common software platform for building and using robots. This common platform permits code and ideas to be more readily shared between contributors. The ROS has been remarkably successful: it con-

tains over 2000 software packages and is written and maintained by almost 600 people [20]. The main advantage the ROS brings to the present research is its structured communication. The topics and messages exchanged between users allows the message flows in the system to be securely maintained. Agents are represented as nodes in the ROS communication system. Each node (called a "topic") can read (listen) and write (publish) over different network connections in the ROS. Figure 2.2 shows an example of a wide ROS graph.

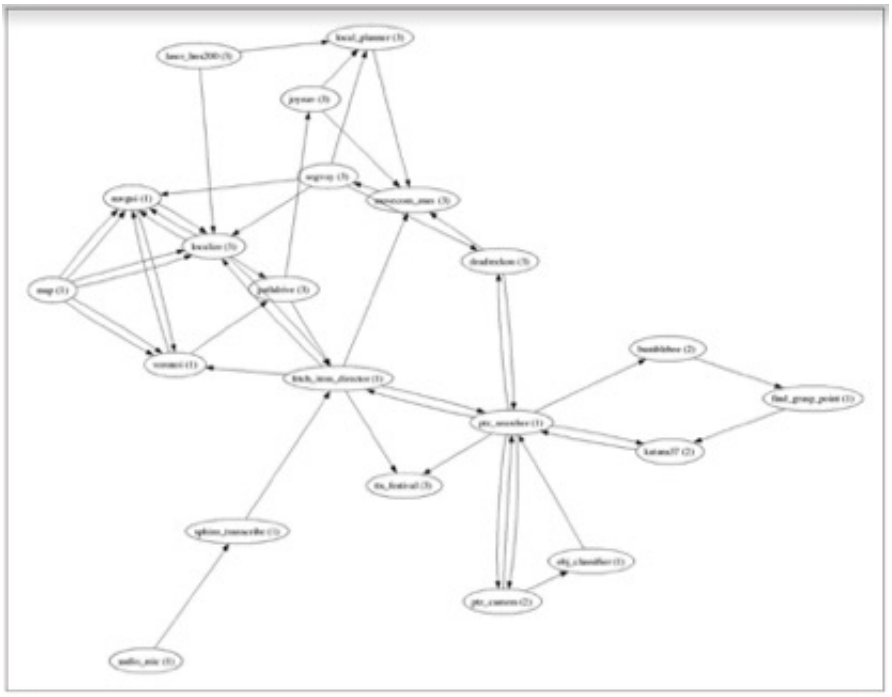


Figure 2.2: Example ROS graph: nodes in the graph represent individual programs; edges represent message streams that communicate sensor data, actuator commands, planner states, intermediate representations, etc.

Another advantage of the ROS is its worldwide community of users, which provides open-source code and sustainable updates for its members. Python

is one of the main ROS client libraries and is strongly supported by the community and relatively simple to use. The community provides a variety of helpful programs that are used herein. For example, Gazebo creates virtual environments for robots that enforce the laws of physics, and Rviz provides graphical user interfaces.

Linux (especially Linux Ubuntu) is the main operating system that supports the ROS. The present work uses the ROS "Indigo" version and Ubuntu version 16.04.

2.6 Necessity for New Hardware

Previous implementations of algorithms to solve DCOP_MST were reported by Yedidsion and suffered from numerous hardware limitations [34]. To address real-world application properties and obstacles, several adjustments were made to the original DCOP_MST model. One of these adjustments resulted in a model called $DCOP\_MST^R$ [33] in which robots have their sensors directed strictly forward (unlike in the original model where agents have a 360° sensing range). An example of such a sensor is the robot front camera, which has a limited field of vision. The iRobots used by Yedidsion proved to be very difficult to work with; for example, they had no on-board computer, so a laptop had to be mounted on each robot. The robots also had no localization or communication capabilities. Addressing these issues clearly required better hardware to simulate realistic DCOP_MST scenarios.

2.7 Hamster Robots

Hamsters (Figure 2.3) are small, robust autonomous robots for research and prototype development. Hamsters can create indoor maps and localize themselves indoors by using 360° LIDAR (LiDar A2M8) and other on-board sensors to provide accurate position information while in motion. They have WiFi ew-7811uac EDIMAX AC600 hardware for wireless communication, a Camera raspberry Pi module v2 located at the front of the robot, and use the ROS. Hamsters include inherent communication and localization features that eliminate localization and communication limitations when positioning these robots. Their on-board Raspberry Pi 3 also dispenses with the requirement for mounted laptops, making the workflow much easier because only a single computer is required for coding and execution.

Figure 2.3: Hamster autonomous robots.

2.8 Creating Simulations & Setting Robots

In this work, the words ”agent” and “robot” are used interchangeably.

2.8.1 Packages

Basic libraries and packages allowing robot control:

1. The `actionlib` library allows movement commands to be sent via `MoveBaseGoal` messages from the `move_base_msgs.msg` library. These messages give the final robot position at the end of movement.
2. We use the `SLAM` algorithm to create a map of the environment. The algorithm is part of built-in packages in Hamster.
3. We use tools from the `map_server` library to save the maps.
4. To localize itself, Hamster uses the augmented Monte Carlo localization algorithm, which is part of the `amcl` package.
5. To create paths in a given environment Hamster uses the `move_base` package to build global and local *costmaps* of the surrounding area. *Costmaps* are special maps used to emphasize obstacles around the robot, thereby helping the robots to avoid the obstacles and creating the shortest and safest path possible.

2.8.2 Setting up the `Gazebo` simulation

A simulation is a safe way to verify the primary code while avoiding the technical complications of real robots. In simulations, the connection to

the robot is stable, and problems with the battery and/or physical damage are avoided. We used the `Gazebo` framework. To create a new world in `Gazebo`, we described *objects* (stored in `.stl` and `.dae` files), physical *characteristics* such as gravity (stored in `.urdf` files), and world *properties* describing the location of each object (stored in `.world` files).

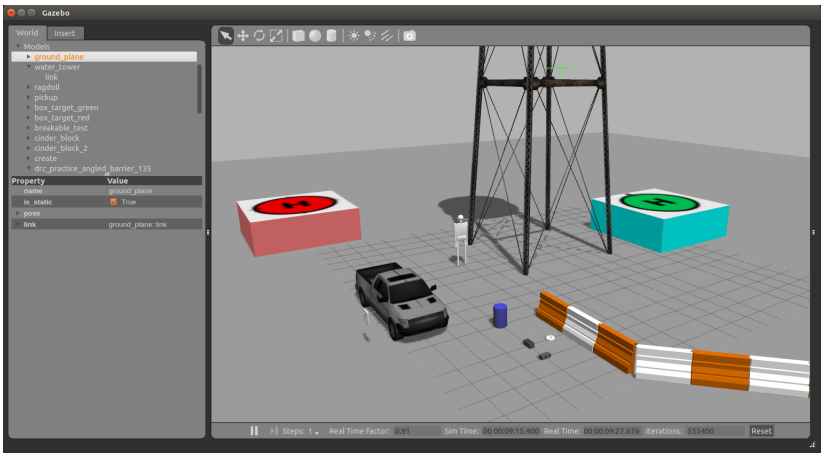


Figure 2.4: Examples of objects in a simulation.

Figure 2.4 shows the variety of shapes, materials, and mechanisms available in this framework. We use a small fraction of these capabilities: robot models, walls, and simple cylinders.

In Gazebo, the Hamster robot model is provided by the company Cogniteam [2]. Figure 2.5 shows the robot model in a simulation. The model reads and writes to the ROS topics exactly the same way as does a real robot. The ROS is indifferent to who is behind the topics, a simulation or a real robot. In both cases the ROS network provides messages regarding relevant topics. This property of the ROS significantly facilitates development and further approaches simulations to real conditions.

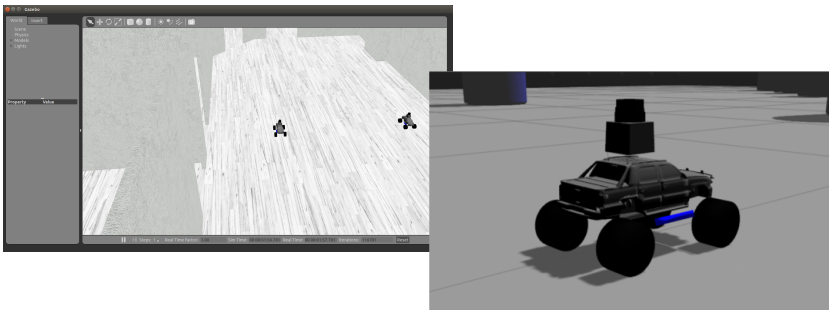


Figure 2.5: Robots in simulations.

2.8.3 Example of the full setting

Figure 2.6 shows a `Gazebo` simulation of a simple experiment. Blue cylinders are the targets and cubes represent the walls. Figures 2.7 and 2.8 show maps of the same experiment made by the `map_server` library for the simulation and in the real world, respectively. Figure 2.9 shows a photograph of the actual robots in the laboratory.

2.8.4 Setting up a Two-Dimensional PyGame Simulation

Rapid testing of new ideas and their verification on a large scale with dozens of robots and targets on the same field requires a simple, small-footprint program. For this purpose, we introduce the *DCOP_MST 2D simulator*. Figure 2.10 shows several screenshots of experiments running on this simulator.

The simulator can run several algorithms simultaneously and allows us to define the size of the field, the number of robots and targets, *mr* and *sr* for the robots, the number of problems to solve per algorithm, and the number of iterations allowed. The user can also play with target requirements and

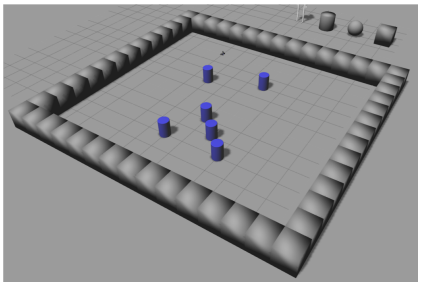


Figure 2.6: (a)

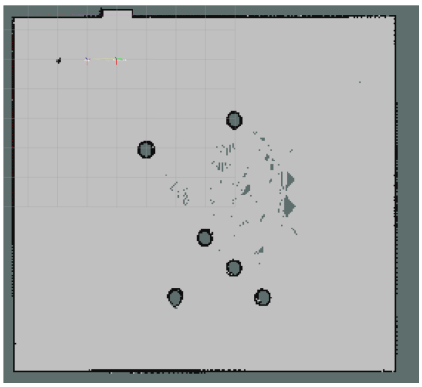


Figure 2.7: (b)

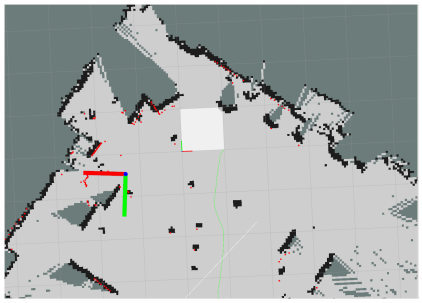
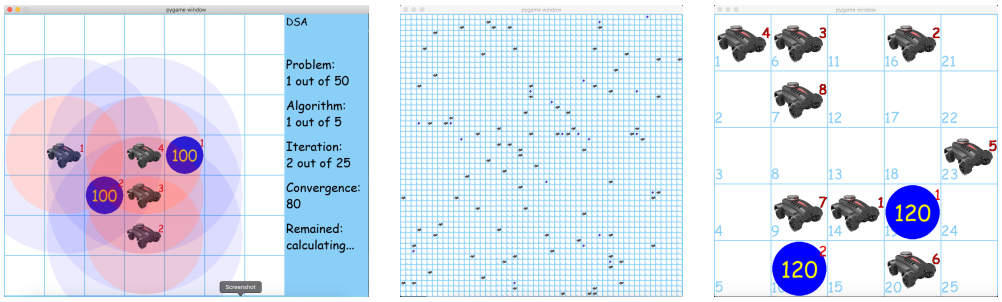


Figure 2.8: (c)

Figure 2.9: (d)

robot credibilities (e.g., making them identical for all robots or giving them unique values). Ranges, the robots' *cred* values, and the targets' *ER* are also allowed to change during the simulation. These capabilities help us better model dynamic, realistic scenarios.

As output, the simulator returns the coverage rate of the algorithms and the number of collisions produced in each iteration.



(a) (b) (c)

Figure 2.10: (a) *sr* and *mr* of the robots. (b) Robots in a large field and (c) in a small field.

3 Collision-Avoiding Max-sum Algorithm

Yedidsion et al. obtained a much better solution quality with the Max-sum_MST exploration algorithm than with the MST distributed local search algorithms, all while limiting the exponential runtime associated with message computation in Max-sum [33]. Their empirical study reveals that Max-sum_MST outperforms all competing algorithms, including DSA_MST and MGM_MST [33]. However, Max-sum_MST does not prevent collisions between mobile sensors, which may result in damage to the sensors, execution delay, or even the inability to perform the coverage task. Figure 3.1 presents an example scenario in which collisions affect MST performance. In this example: red dots are targets, colored crosses are the locations available to the robots, and highlighted circles show possible collisions due to the intersection of robot paths. To avoid collisions, we propose the Collision-Avoiding Max-sum (CAMS) algorithm, a version of the Max-sum algorithm that allows agents to select the deployment that maximizes coverage and yet avoid collisions. This is achieved by adding function-nodes representing locations to which the agents can move to the factor graphs generated in each iteration of the algorithm (before each

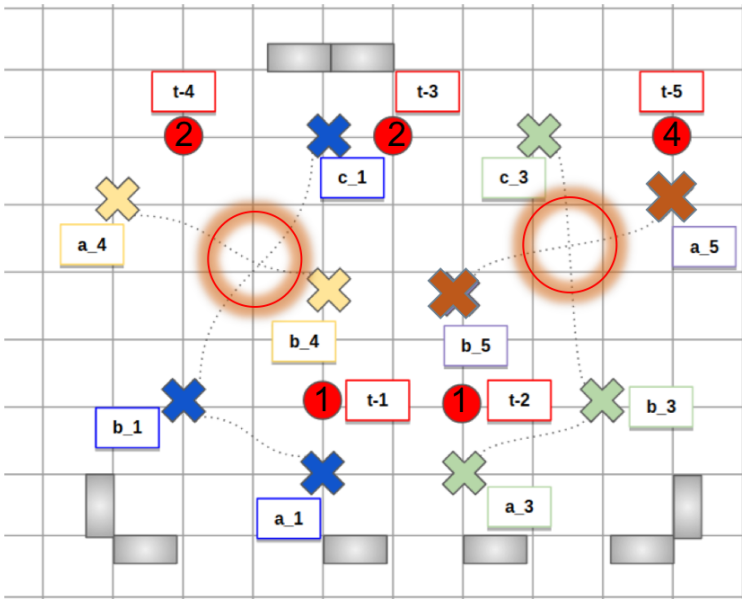


Figure 3.1: Path $a_4 \rightarrow b_4$ intersects with $c_1 \rightarrow b_1$ and path $b_5 \rightarrow a_5$ intersects with $c_3 \rightarrow b_3$, indicating possible collisions between robots.

movement of the agents). Each function-node can represent either locations to which only a single agent can move or locations to which two agents can move.

The first type of function-node assigns zero (positive) utility to the option in which the agent chooses not to (to) move to the given location.

The second type of function-node represents a hard constraint that prevents two agents from selecting the same location. In this case, its utility function imposes zero utility for when neither agent moves to the given location, positive utility when only one agent moves to the location, and $-\infty$ utility when both agents move to the location.

If more than two agents can select a location, the agents generate function-nodes that represent the binary constraint between each pair of agents.

More formally, a factor graph generated by agents in CAMS contains three

types of function-nodes:

I. FT_j: This type of function-node represents a target T_j. As in Max-sum_MST, the utility that agent A_i derives for covering T_j is $\min\{ER_{FT_j}, cred_i\}$. However, in nondegenerate cases where multiple sensors are required for coverage, the utility derived is $cred_i$.

II. $FL_{(i,e)}$: This type of function-node represents a location l to which agents A_i and A_e can move in the given iteration. The utility is zero if neither agent selects location l, $-\infty$ if both agents select l, and a random utility is selected from a range of numbers all much smaller than ER_{FT_j} if only one of them selects l,[1] Although scenarios may exist in which more than two agents can move to the same location, FL is a binary constraint, so, if $k > 2$ agents can select the same location, a function-node FL is assigned to each pair of the k agents.

III. $FL_{(i)}$: This type of function-node represents a location to which only a single agent can move. In this case, the corresponding constraint is unary. The utility is a random positive number if A_i selects this location (the utility is selected from the same range as the random positive utilities selected for the binary constraints) and zero if A_i does not select this location.

Figure 3.2 presents an example of a factor graph generated in an arbitrary iteration of CAMS. It includes two mobile sensors, each with four possible locations to which to move (up, down, left, and right) and the option to stay in its current location. All function-nodes representing locations to which only one mobile sensor can move are of type III. Although the

[1] Random numbers are selected to avoid ties between desired options, as per Ref. [7].

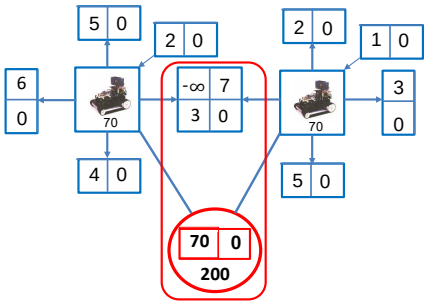


Figure 3.2: A factor graph generated in CAMS.

domain of each agent includes five values (representing the possible locations the agent can select), the utility is positive only if the agent selects the location represented by the function-node; for all other locations the utility is zero. The middle location (to which both mobile sensors can move) is represented by a type-II function-node. It includes four options: neither agent selects this location (zero utility), both agents select this location ($-\infty$ utility), and one or the other agent selects this location (positive utility). The target is represented by a type-I function-node. Its coverage requirement is 200, whereas the credibility of each mobile sensor is 70, which is the utility derived for covering the target. In this example, covering the target is only possible from the middle location (to which both mobile sensors can move). However, the agents collide if they both move to this location.

Compared with Max-sum_MST, the overhead runtime complexity of CAMS is negligible because the additional function-nodes representing unary and binary constraints require at most 2^2 utility comparisons for each message produced. Conversely, the comparisons required for generating a message by each function-node FT that represents a target (in both algorithms) is 2^k, where k is the number of neighbors of FT. However, we expect that adding function-nodes representing locations will result in more cycles

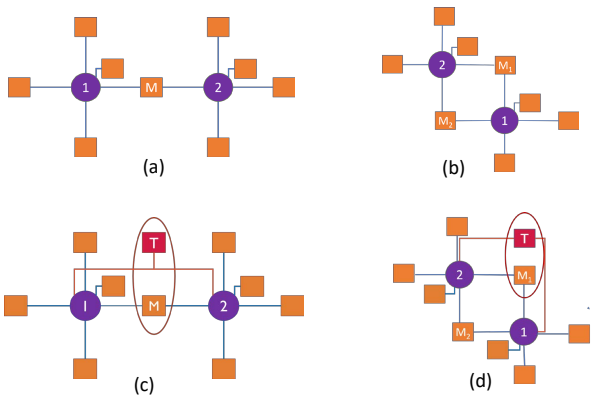


Figure 3.3: Scenarios with convergence guarantees.

and thereby reduce the probability for convergence. We demonstrate in the following sections that this is not the case for the CAMS algorithm.

3.1 Analyzing Specific Scenarios

We next present a set of scenarios for which we prove the convergence of CAMS to a noncolliding (optimal) state. The following section presents a second set containing scenarios for which our empirical results demonstrate such convergence.

Figure 3.3 shows scenarios for which we have established convergence guarantees. Below, we prove that the CAMS algorithm converges to collision-free optimal solutions when applied to these scenarios.

Proposition 1 *Max-sum converges in a linear number of steps to a collision-free optimal solution when solving the scenario depicted in Figure 3.3(a).*

Proof: The factor-graph representation of this scenario has a tree structure so the algorithm will converge in a linear number of steps to an optimal

solution [40]. This optimal solution cannot include the selection by both agents of the mutual location M since its cost is $-\infty$. □

Proposition 2 *When solving the scenario depicted in Figure 3.3(b), Max-sum converges to a collision-free optimal solution in a pseudo-linear number of steps.*

Proof: The factor-graph representation of this scenario includes a single cycle with two function-nodes. Each function-node has four entries in its utility table. According to Ref. [8], when belief propagation is applied to a single cycle graph, Max-sum converges to the optimal solution if and only if the optimal repeated path is consistent. The only way to generate an inconsistent path in such a cycle with two function-nodes and a four-entry utility table is if the cycle includes opposing directed diagonals in the utility tables. However, in our case, one of these diagonals must include the $-\infty$ entry. Thus, the maximal path must be consistent. The number of steps depends on the constant utilities sent by the neighbors of the unary function-node, which are not included in the cycle. If these utilities differ only negligibly, the convergence time is linear (i.e., on the order of the cycle size). □

The next lemma is relevant to scenarios (as depicted in Figure 3.2) in which their representing factor graph includes a single-target function-node and a single location from which the target can be covered and to which both agents can move.

Lemma 1 *When Max-sum operates on factor graphs as described above, the target function-node FT sends the same messages to its two neighbors MS_1 and MS_2 in every step, including zero for not covering the target and $cred_1$ or $cred_2$, respectively, for covering the locations (assuming $ER_{FT} > \max(cred_1, cred_2)$).*[2]

[2]A previous indication that hard constraints reduce the complexity of a factor-graph

Proof: We follow the path of messages over a cycle, including one target function-node FT, two mobile sensors MS_1 and MS_2, and one mutual location ML to which both sensors can move. We denote by c_1 and c_2 the utilities included in FL_{ML} (besides zero and $-\infty$) for the options that only MS_1 or only MS_2 move to ML. Recall that, by construction, $cred_1$, $cred_2 \gg c_1$, c_2. Figure 3.4 shows a factor graph that applies to this description.

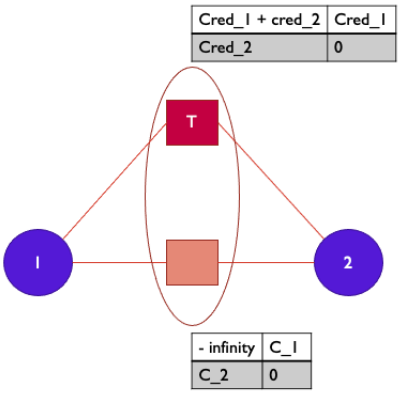


Figure 3.4: Target, two robots, and one common cell.

Without loss of generality, we follow the cyclic message path starting at FT, then proceed to MS_1, ML, MS_2, and then back to FT:

1. The first message $FT \rightarrow MS_1$ includes the pair $\langle 0, cred_1 \rangle$ (the left and right entries of the pair represent the belief in not covering and covering the target, respectively).

2. The second message $MS_1 \rightarrow FL_{ML}$ includes the same pair $\langle 0, cred_1 \rangle$ because the message from FT is the only message requiring further transmission.

representation of a realistic application (although very different from the case this lemma analyzes) is available in Ref. [18].

3. The third message $FL_{ML} \rightarrow MS_2$ includes

$$\langle \max\{c_2, -\infty\},\ \max\{0, cred_1 + c_1\} \rangle.$$

Thus, the message includes $\langle c_2, cred_1 + c1 \rangle$.

4. The fourth message $MS_2 \rightarrow FT$ passes the same $\langle c_2, cred_1 + c1 \rangle$.

5. The next $FL \rightarrow MS_1$ message includes

$$\langle \max\{cred_2 + c_2, cred_1 + c_1\},\ \max\{cred_1 + cred_2 + c_2, 2cred_1 + c_1\} \rangle.$$

In both cases, regardless of whether $cred_2 + c_2 > cred_1 + c_1$, the message will be normalized down to $\langle 0, cred_1 \rangle$.

We now follow the cyclic message in the opposite order of nodes starting at FT, then MS_2, ML, MS_1, and then back to FT:

1. The first message $FT \rightarrow MS_2$ includes the pair $\langle 0, cred_2 \rangle$ (the left and right entries of the pair represent the beliefs in not covering and covering the target, respectively).

2. The second message $MS_2 \rightarrow FL_{ML}$ includes the same pair $\langle 0, cred_2 \rangle$ because the message from FT is the only message requiring further transmission.

3. The third message $FL_{ML} \rightarrow MS_1$ includes

$$\langle \max\{c_1, -\infty\},\ \max\{cred_2 + c_2, 0\} \rangle.$$

Thus, the message includes $\langle c_1, cred_2 + c_2 \rangle$.

4. The fourth message $MS_1 \rightarrow FT$ passes the same beliefs $\langle c_1, cred_2 + c_2 \rangle$.

5. The next message $FL \rightarrow MS_2$ includes

   $$\langle \max\{cred_1 + c_1, cred_2 + c_2\},\ \max\{cred_1 + cred_2 + c_1, 2cred_2 + c_2\} \rangle.$$

   In both cases here as well, regardless of whether $cred_2 + c_2 > cred_1 + c_1$, the message will be normalized down to $\langle 0, cred_2 \rangle$.

A similar analysis applies to any case where $cred_1,\ cred_2 \gg c_1,\ c_2$. □

Intuitively, this happens because the $-\infty$ utility in ML prevents the utility calculation including $cred_i$ from being sent to MS_j for the covering option, where $i \neq j$. Thus, $cred_i$ in the message from MS_j to FT shifts from the covering option to the non-covering option, which offsets the utility added by FT in each cycle.

Proposition 3 *When solving the scenario depicted in Figure 3.3(c), Max-sum converges to a collision-free optimal solution after a linear number of steps.*

Proof: According to Lemma 1, the target representing the function-node in this scenario consistently sends the same messages. Thus, although this scenario includes a single cycle, in practice, the algorithm behaves as if it were solving a tree. □

An immediate corollary is that Max-sum converges in scenarios similar to that depicted in Figure 3.3(c), in which the target can be covered from additional locations because the graph retains a single degenerate cycle (i.e., the algorithm performs as if it were solving a tree-structured factor graph).

To analyze the convergence of the scenario represented by the factor graph depicted in Figure 3.3(d), we need the following lemma:

Lemma 2 *When Max-sum operates on factor graphs as depicted in Figure 3.3(d), the target function-node FT sends the same messages to its two neighbors MS_1 and MS_2 in every step, including zero for not covering the target and $cred_1$ or $cred_2$, respectively, for covering locations [assuming $ER_{FT} > \max(cred_1, cred_2)$].*

Proof: The proof is similar to that of Lemma 1. We follow the path of messages in a cycle, including one target function-node FT, two mobile sensors MS_1 and MS_2, and two mutual locations ML_1 and ML_2 to which both mobile sensors can move. We denote by c_1 and c_2 the utilities included in FL_{ML_1} and by c_3 and c_4 the utilities included in FL_{ML_2} (besides zero and $-\infty$), for the options that only MS_1 moves to ML_1 or only MS_2 moves to ML_2. Recall that, by construction, $cred_1$, $cred_2 \gg c_1, c_2, c_3, c_4$. Figure 3.5 shows a factor graph that applies to this description.

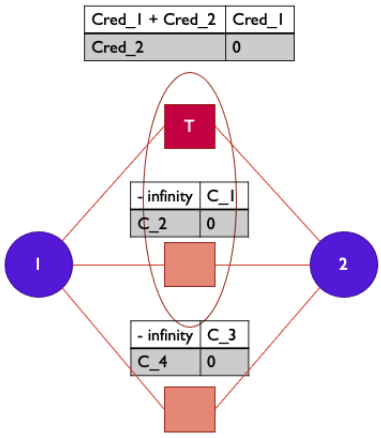


Figure 3.5: Target, two robots, and two common cells.

Without loss of generality, we follow the cyclic message path starting at FT, then MS_1, ML_1 and ML_2, MS_2, and then back to FT (the left and right entries of the pair represent the beliefs that ML_1 covers the target and that ML_2 does not cover the target, respectively):

1. Step 1:

$FT \rightarrow MS_1$ includes the pair $\langle cred_1, 0\rangle$.

$ML_1 \rightarrow MS_1$ includes the pair $\langle c_1, c_2\rangle$.

$ML_2 \rightarrow MS_1$ includes the pair $\langle c_3, c_4\rangle$.

2. Step 2:

   $MS_1 \rightarrow ML_1$ includes the pair $\langle cred_1 + c_3, c_4\rangle$.

   $MS_1 \rightarrow ML_2$ includes the pair $\langle cred_1 + c_1, c_2\rangle$.

3. Step 3:

   $ML_1 \rightarrow MS_2$ includes $\langle c_2 + c_4, cred_1 + c_1 + c_3\rangle$.

   $ML_2 \rightarrow MS_2$ includes $\langle 0, c_4\rangle$.

4. Step 4:

   $MS_2 \rightarrow FT$ includes $\langle c_2, cred_1 + c1 + c_3\rangle$.

   We denote c_2 as α and $c1 + c_3$ as β.

5. The next message $FL \rightarrow MS_1$ includes

   $$\langle \max\{cred_1 + cred_2 + \alpha, 2cred_1 + \beta\},\ \max\{cred_2 + \alpha, cred_1 + \beta\}\rangle.$$

   In both cases, regardless of whether $cred_2 + \alpha > cred_1 + \beta$, the message is normalized to $\langle cred_1, 0\rangle$.

A similar analysis applies to the opposite order of nodes, where $cred_1$, $cred_2 \gg c_1$, c_2, c_3, c_4. □

Proposition 4 *When solving the scenario depicted in Figure 3.3(d), Max-sum converges to a collision-free optimal solution in a pseudo-linear number of steps.*

Proof: This proof is similar to that of Proposition 3. According to Lemma 2, the target representing the function-node in this scenario consistently sends the same messages. Thus, although this scenario includes two cycles, in practice, the algorithm behaves as if it were solving a single-cycle factor graph. This cycle cannot include an inconsistent optimal path (for the same reason as given in Proposition 2), so the algorithm converges to the optimal solution, which cannot include collisions. We omit the similar argument proving that a pseudo-linear number of steps is required for convergence. □

3.2 Experimental Evaluation

To evaluate the performance of CAMS, we designed two types of simulation environments. The first was a software simulation implemented in Python (see Chapter 2.8) and the second was a simulation involving Hamster robots [2]. We start by evaluating the performance of Max-sum on small scenarios, including at most two targets for which we were unable to establish guaranteed convergence (see Figure 3.6).

For each scenario we produced 50 instances. In each scenario, the target's ER was 120 and the credibility of each sensor was 30. The positive utilities of the location function-nodes were selected randomly between 1 and $1M$ and divided by $10\,000M$, resulting in random numbers in the range $[0.000\,000\,000\,01,\ 0.0001)$. An empirical evaluation reveals that Max-sum always converges to an optimal collision-free solution when solving these scenarios. Beneath each scenario, the average number of iterations required for convergence (on the left) and to obtain the standard deviation (on the right) are given in brackets.

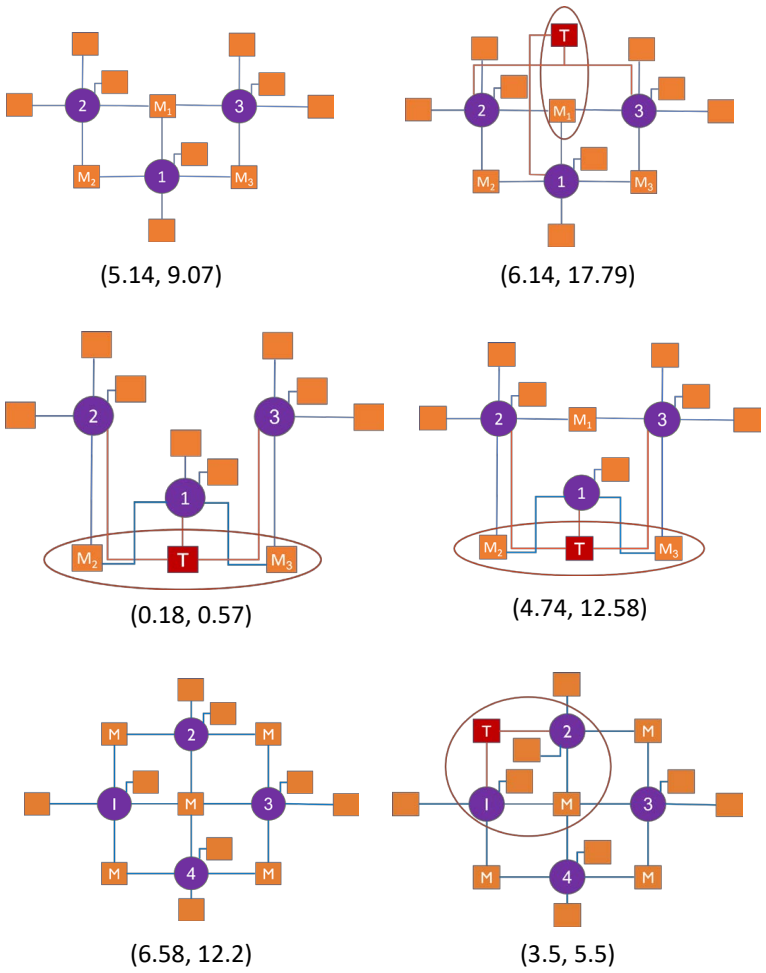
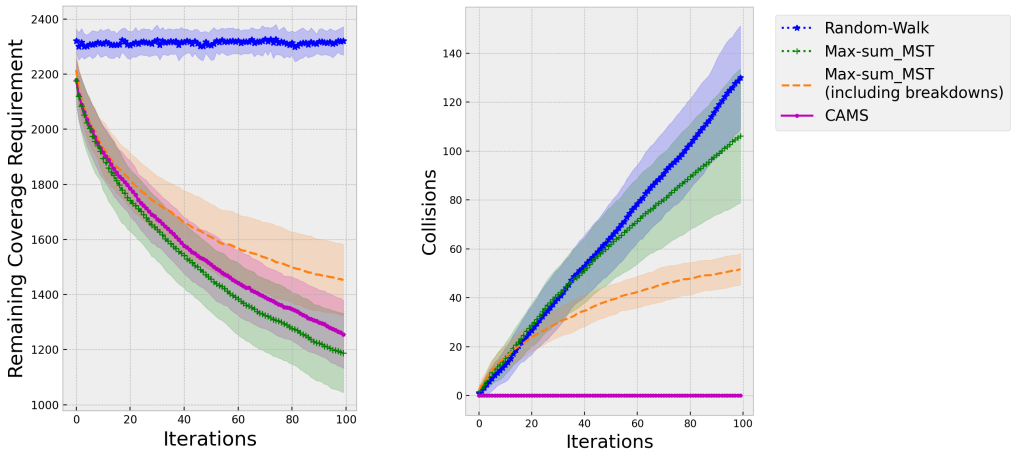


Figure 3.6: Scenarios with no convergence guarantees.

Next, we generate larger scenarios in which 20 targets were randomly positioned on a grid with dimensions 50×50. Eighty mobile sensors are also positioned randomly on this grid, such that every cell of the grid includes at most one mobile sensor. The targets' *ER*, the credibility of mobile sensors, and the utilities for the cell representing function-nodes are selected in the same way as done for the small scenarios described above. In each iteration of the algorithm, each mobile sensor can either move to one of the four adjacent cells (up, down left or right) or stay in its location. We

assumed that a mobile sensor can cover a target located in the cells closest to it in each direction, including row, column, and diagonals (one step in each direction).

We compare CAMS with Max-sum_MST and with a random walk algorithm. Each algorithm performs 100 iterations in which the mobile sensors select locations. CAMS and Max-sum_MST perform 30 steps of the algorithm in each iteration before the agents select their locations. The remaining coverage in each iteration is calculated by using $\sum_{T_j \in T} cr(T_j)$. The group goal is thus to minimize the remaining coverage requirement.



(a) (b)

Figure 3.7: (a) Remaining coverage and (b) accumulated collisions as a function of number of iterations.

Figure 3.7(a) presents the remaining coverage requirements for the three algorithms as a function of number of iterations (colors represent the standard deviation). For Max-sum_MST we included the results of two experiments. In the first experiment, the mobile sensor movements are not affected by collisions. In the second experiment, colliding sensors break

down and stop moving. CAMS clearly maintains a level of coverage similar to that of Max-sum_MST and both have a large advantage over the random walk algorithm. However, since the mobile senors in Max-sum_MST do not avoid collisions, the agents performing it are less restricted, so the resulting coverage is slightly better than the coverage achieved by CAMS (the differences are insignificant with $p = 0.01$). Conversely, CAMS significantly outperforms Max-sum_MST in scenarios that include coverage breakdowns. Figure 3.7(b) shows the number of accumulated collisions for each algorithm as a function of number of iterations. CAMS is collision free whereas random walk and Max-sum_MST experience collisions at a similar rate.

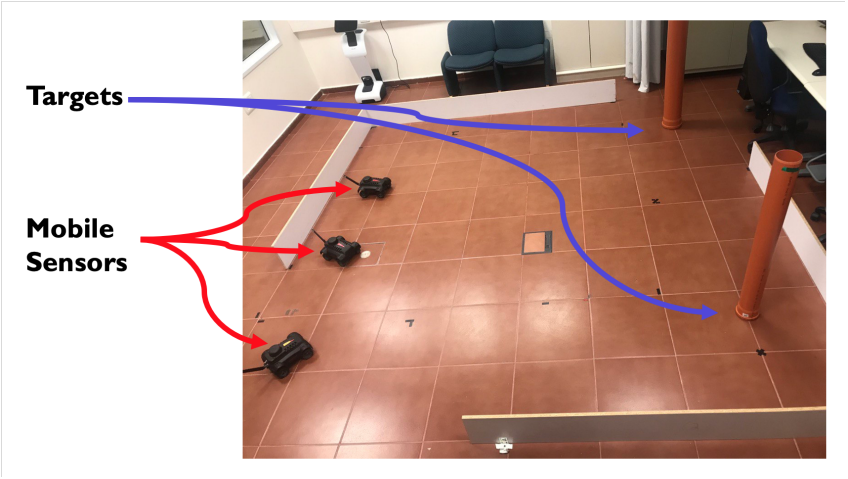


Figure 3.8: Experimental setup.

The goal of the next set of experiments was to examine the delay caused by collisions between robots in realistic settings. Thus, the experiments included a MST composed of three Hamster robots [2] and two targets with $ER = 60$ placed in a 4×4 grid, with each grid cell measuring a square meter. The targets were placed randomly in nonadjacent cells of the grid. The Max-sum algorithm executed the same number of steps and had the

same credibility in each iteration as all other experiments presented in this section. The algorithms executed ten iterations. We selected four different target positions and, for each target position, five different robot positions, resulting in 20 experiments for each algorithm.[3] Figure 3.8 shows the initial position of the Hamsters for one of the experiments.

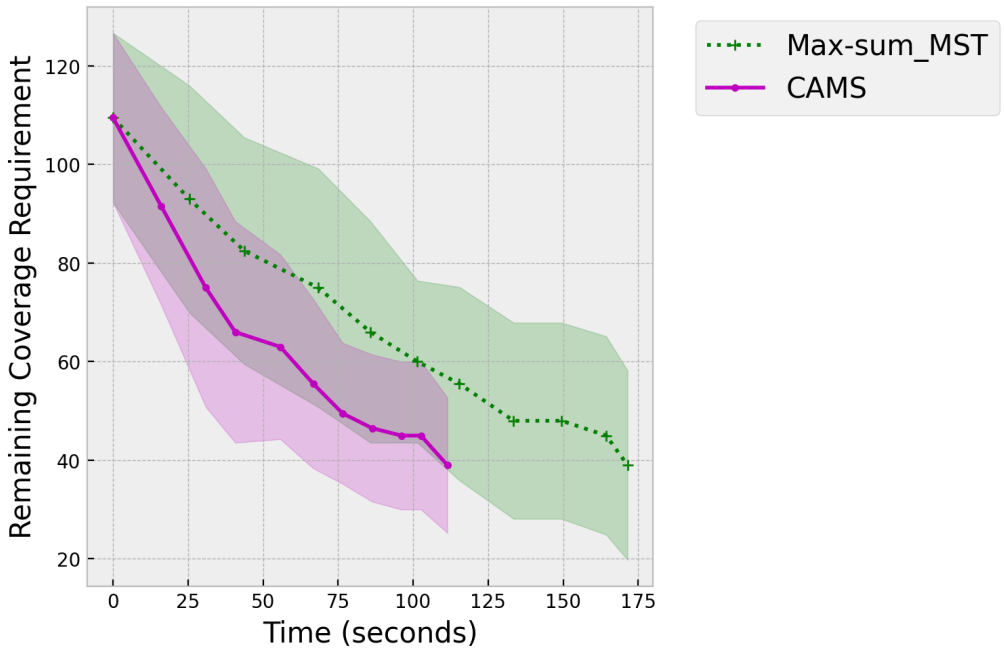


Figure 3.9: Remaining coverage requirement as a function of experiment execution time for Max-sum_MST and CAMS algorithms.

Figure 3.9 shows the remaining coverage requirement as a function of experiment execution time for the Max-sum_MST and CAMS algorithms. Although both algorithms produce the same level of coverage after completing ten iterations, CAMS avoids collisions and thus reaches this coverage

[3] A video presenting the highlights of this set of experiments is included in the supplemental material.

state roughly 36% faster.

4 Conclusions

Realistic multi-agent applications often include scenarios in which a mobile sensor team (MST) must detect and monitor targets. DCOP_MST is a dynamic DCOP formulation that we use to model such MST scenarios and that allows us to use various algorithms to solve them.

An important aspect of applications that include mobile sensors is that the sensors should avoid collisions while optimizing coverage. To achieve this challenging combination, we propose the CAMS algorithm, which is an offshoot of the Max-sum_MST algorithm that avoids collisions by adding to the factor-graph representation of the problem (in addition to function-nodes representing targets) hard-constraint function-nodes representing the locations to which agents may choose to move. In contrast with expectations, this addition does not prevent the algorithm from converging. The analysis of simple scenarios offers explanations for this phenomenon, and the empirical results reveal that the desired properties are maintained upon scaling up the problem. Finally, small scenario experiments with hardware robots reveal the actual delay in execution caused by collisions.

5 Future Work

The work presented in the previous chapters was a first step in our journey towards our research goal, which is to identify and address challenges in applying distributed optimization models and algorithms to real world applications that include mobile sensor teams. We outline several objectives for future research and how we intend to achieve these objectives. It will include theoretical and empirical algorithmic analysis.

5.1 Research Objectives

5.1.1 Collisions in MST

For now, robots move one cell at each iteration. Although there are no collisions during the run, it takes more time to get to the final positions. Allowing robots to move in a greater range expose chances for more collisions. Several algorithms were developed to prevent collisions in multi-agent path planning [12, 22, 26, 32, 37]. We are willing to apply those under DCOP_MST formulation.

We also plan to make better usage of the sensors installed in the Hamster and may add image processing combined with distance measuring

for better positioning.

5.1.2 Communication Disturbance in MST

Communication disturbance can generate substantial interference in realistic applications that include multiple agents aiming to solve a distributed combinatorial optimization problem. Such applications are exceedingly challenging in the presence of other properties of the system, e.g., the need to optimize a global objective goal, while decentralized coordination is constrained by unpredictable dynamic environment [21].

Most studies that investigate the DCOP and DCOP_MST models and algorithms assume that messages arrive instantaneously and that there is no loss of messages. Unfortunately, many of the algorithms proposed for solving the models take advantage of these simplistic assumptions and even depend on them for achieving some desired properties, such as convergence [21, 33, 36].

In our research, we are willing to focus on latency awareness by designing a latency-aware DCOP_MST model that represents the possibly dynamically changing uncertainty in the latency on messages exchanged. We plan to investigate the performance of existing DCOP_MST algorithms in the presence of message latency and propose some new asynchronous distributed DCOP_MST algorithms.

5.1.3 Learning *EnvironmentalRequirement* in MST

Recent studies have investigated how a team of mobile sensors can cope with real world constraints, such as uncertainty in the performance, dy-

namically appearing and disappearing targets, technology failures and changes in the environment conditions [13, 15, 27, 34]. Lisy et al. proposed an additional element, deception by an adversary, which is relevant in many (military) applications [11]. The adversary is expected to use deception to prevent the sensor team from performing its tasks and can deceptively change the importance that agents give to targets in the area. The opponent is expected to use camouflage in order to create confusion among the sensors regarding the importance of targets, and reduce the team's efficiency in target coverage. In these cases our assumption in DCOP_MST about known ER (Environmental Requirement) is wrong.

In our research, we are willing to enhance DCOP_MST model for such scenarios by applying learning methods of the ER.

5.1.4 MARL in MST

We discussed DCOP formulation to model dynamically changing multi-agent coordination problems, where a dynamic DCOP is a sequence of static DCOPs, each partially different from the DCOP preceding it. We assume that the problem in each time step is decoupled from the problems in other time steps, which might be false assumption in some applications. Moreover, running DCOP algorithms for each action selection through the whole system results in significant communication among agents, which is not practical for most applications with limited communication bandwidth. Coordinated multi-agent reinforcement learning (MARL) provides a promising approach to scaling learning in large cooperative multi-agent systems. DCOP techniques can be used to coordinate action selection among agents during both the learning phase and the policy execution

phase to ensure good overall system performance. [38]

Researchers have introduced the following contributions: (i) Introduction of a new model, called Markovian Dynamic DCOPs (MD-DCOPs), where the DCOP in the next time step is a function of the value assignments in the current time step[16]. (ii) Introduction of distributed reinforcement learning algorithms, that balance exploration and exploitation to solve MD-DCOPs in an online manner. [16, 39] (iii) Development of a learning approach that generalizes previous coordinated MARL approaches that use DCOP algorithms and enables MARL to be conducted over a spectrum from independent learning (without communication) to fully coordinated learning depending on agents' communication bandwidth. [38]

DCOP_MST formulation have a great similarities to those contributions. We are willing to adopt some MARL approaches to increase performance in robot allocation problems.

5.1.5 Breakdowns in MST

In this work we made an assumption that amount of agents is constant during the run of algorithms. However, many real world scenarios have highly dynamic nature in this sense [13, 15, 27, 34]. The agents can break down in different times and their joint capabilities can change dramatically as well. We are willing to explore the performance of existing DCOP_MST algorithms in such challenging environments.

5.2 Expected Advancement

The main statement of this research proposition is the formalization of communication architectures in different domains inside multi agent optimization models and the design of algorithms for solving them. To foster the acceptance of a new model within the scientific community, we will provide algorithmic approaches, proof-of-concept implementations in simulations and in real-world scenarios, and rigid evaluations of significant benchmark instances. We will conduct theoretical evaluations for all the proposed models and algorithms (e.g. time complexity of the proposed model, quality and convergence guarantee of the proposed algorithms, communication costs, etc.) to better characterize and explore their applicability.

We anticipate that the advances we will make in the the research and its associated application areas will have a significant impact on the development of distributed algorithms for more realistic distributed systems. To this end we intend to implement and examine the knowledge acquired in this research on a realistic task allocation application. In addition, we intend to make a set of DCOP_MST benchmark problems available to the public, to foster future research in the direction of the proposed work.

6 Declaration

I hereby confirm that this thesis is entirely my own work. I confirm that no part of the document has been copied from either a book or any other source—including the internet—except where such sections are clearly and correctly identified within the text or in the bibliography.

Bibliography

[1] Z. Chen, Y. Deng, T. Wu, and Z. He. A class of iterative refined max-sum algorithms via non-consecutive value propagation strategies. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 32(6):822–860, 2018.

[2] Cogniteam. *Hamster V7 Smart ROS Autonomous Ground Vehicles for Industry and Academic R&D*, 2020.

[3] Y. Deng and B. An. Speeding up incomplete gdl-based algorithms for multi-agent optimization with dense local utilities. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 31–38, 2020.

[4] Farinelli, Rogers, Petcu, and Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS*, 2008.

[5] A. Farinelli, A. Rogers, and N. Jennings. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *JAAMAS*, 2013.

[6] A. Farinelli, A. Rogers, and N. R. Jennings. Agent-based decentralised coordination for sensor networks using the max-sum algo-

rithm. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 28(3):337–380, 2014.

[7] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceeding of the 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 639–646, 2008.

[8] G. D. Forney, F. R. Kschischang, B. Marcus, and S. Tuncel. Iterative decoding of tail-biting trellises and connections with symbolic dynamics. In B. Marcus and J. Rosenthal, editors, *Codes, Systems, and Graphical Models*, pages 239–264. Springer, 2001.

[9] M. Jain, M. E. Taylor, M. Yokoo, and M. Tambe. Dcops meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 181–186, 2009.

[10] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:2:181–208, February 2001.

[11] V. Lisý, R. Zivan, K. P. Sycara, and M. Pechoucek. Deception in networks of mobile sensing agents. In *AAMAS*, pages 1031–1038, 2010.

[12] R. Luna and K. E. Bekris. Network-guided multi-robot path planning in discrete representations. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4596–4602, 2010.

[13] K. S. Macarthur, R. Stranders, S. D. Ramchurn, and N. R. Jennings. A distributed anytime algorithm for dynamic task allocation in multi-

agent systems. In *Proceedings of the 25th Conference of the American Association for Artificial Intelligence (AAAI)*, 2011.

[14] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraints optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.

[15] D. T. Nguyen, W. Yeoh, H. C. Lau, S. Zilberstein, and C. Zhang. Decentralized multi-agent reinforcement learning in average-reward dynamic DCOPs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1447–1455, 2014.

[16] D. T. Nguyen, W. Yeoh, H. C. Lau, S. Zilberstein, and C. Zhang. Decentralized multi-agent reinforcement learning in average-reward dynamic dcops. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, page 1447–1455. AAAI Press, 2014.

[17] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[18] T. Penya-Alba, M. Vinyals, J. Cerquides, and J. A. Rodríguez-Aguilar. A scalable message-passing algorithm for supply chain formation. In *Proceedings of the 26th Conference of the American Association for Artificial Intelligence (AAAI)*, 2012.

[19] M. Pujol-Gonzalez, J. Cerquides, P. Meseguer, J. A. Rodríguez-Aguilar, and M. Tambe. Engineering the decentralized coordination of uavs with limited communication range. In *Advances in Artificial Intelligence*, pages 199–208. Springer, 2013.

[20] M. Quigley, B. Gerkey, and W. D. Smart. *Programming Robots with ROS*. O'Reilly Media, 2015.

[21] B. Rachmut, R. Zivan, and W. Yeoh. Latency-aware localsearch for distributed constraint optimization. In *IFAAMAS*, 05 2021.

[22] V. Rahmani and N. Pelechano. Multi-agent parallel hierarchical path finding in navigation meshes (ma-hna*). *Computers & Graphics*, 86, 11 2019.

[23] S. D. Ramchurn, A. Farinelli, K. S. Macarthur, and N. R. Jennings. Decentralized coordination in robocup rescue. *Computer Journal*, 53(9):1447–1461, 2010.

[24] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 2011.

[25] P. Rust, G. Picard, and F. Ramparany. Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 468–474, 2016.

[26] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195:470 – 495, 2013.

[27] M. C. Silaghi, D. Sam-Haroud, M. Calisti, and B. Faltings. Generalized english auctions by relaxation in dynamic distributed csps with private constraints. In *IJCAI-01 DCR Workshop*, Seattle, 2001.

[28] R. Stranders, F. M. Delle-Fave, A. Rogers, and N. R. Jennings. A decentralised coordination algorithm for mobile sensors. In *AAAI*, 2010.

[29] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised coordination of mobile sensors using the max-sum algorithm. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 299–304, 2009.

[30] W. T. L. Teacy, A. Farinelli, N. J. Grabham, P. Padhy, A. Rogers, and N. R. Jennings. Max-sum decentralised coordination for sensor systems. In *AAMAS*, pages 1697–1698. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[31] G. Wang, G. Cao, P. Berman, and T. F. Laporta. A bidding protocol for deploying mobile sensors. In *Proceedings of the 11th IEEE International Conference on Network Protocols (IEEE ICNP)*, 2003.

[32] K.-H. C. Wang and A. Botea. Tractable multi-agent path planning on grid maps. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, page 1870–1875, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[33] H. Yedidsion. *Distributed Constraint Optimization for Teams of Mobile Agents*. PhD thesis, Ben Gurion University, 2015.

[34] H. Yedidsion and R. Zivan. Applying dcop_mst to a team of mobile robots with directional sensing abilities (extended abstract). In *AAMAS*, 2016.

[35] H. Yedidsion, R. Zivan, and A. Farinelli. Applying max-sum to teams of mobile sensing agents. *Engineering Applications of Artificial Intelligence (EAAI)*, 71:87–99, 2018.

[36] H. Yedidsion, R. Zivan, and A. Farinelli. Applying max_sum to teams of mobile sensing agents. In *EAAI*, 2018.

[37] B. Zahy, S. Roni, F. Ariel, Z. Roie, and O. Steven. Multi-agent path finding for self interested agents. *Proceedings of the 6th Annual Symposium on Combinatorial Search, SoCS*, 01 2013.

[38] C. Zhang and V. Lesser. Coordinating multi-agent reinforcement learning with limited communication. volume 2, pages 1101–1108, 05 2013.

[39] Z. Zhang, D. Zhao, J. Gao, D. Wang, and Y. Dai. Fmrq—a multiagent reinforcement learning algorithm for fully cooperative tasks. *IEEE Transactions on Cybernetics*, 47(6):1367–1379, 2017.

[40] R. Zivan, O. Lev, and R. Galiki. Beyond trees: Analysis and convergence of belief propagation in graphs with multiple cycles. In *Proceedings of the 34th International Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, pages 7333–7340, 2020.

[41] R. Zivan, S. Okamoto, T. Parash, L. Cohen, and H. Peled. Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. In *JAAMAS*, 2017.

[42] R. Zivan, T. Parash, L. Cohen, H. Peled, and S. Okamoto. Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 31(5):1165–1207, 2017.

[43] R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, and K. Sycara. Distributed constraint optimization for teams of mobile sensing agents. In *JAAMAS*, 2015.

[44] R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, and K. P. Sycara. Distributed constraint optimization for teams of mobile sensing agents.

Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS), 29(3):495–536, 2015.