



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**

February 2021



אוניברסיטת בן-גוריון בנגב
הפקולטה למדעי ההנדסה
המחלקה למהנדסת תעשייה וניהול

אתגרי אופטימיזציה מבוזרת באפליקציות מרובות סוכנים

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



Coordinating a Mobile Sensor Team (MST) to cover targets is a challenging problem in many multiagent applications. Such applications are inherently dynamic due to changes in the environment, technology failures, and incomplete knowledge of the agents. Agents must adaptively respond by changing their locations to continually optimize the coverage of targets. Distributed Constraint Optimization Problem (DCOP) is a general framework for describing distributed problems including constraints, which can be represented by a graphical model and solved using message passing algorithms. Recently, a variation of the DCOP model was adjusted for representing such problems including Mobile Sensor Teams (DCOP_MST) and incomplete algorithms, such as Max-sum, were enhanced with exploration methods in order to solve them. In DCOP_MST, agents maintain variables for their physical positions, while each target is represented by a constraint that reflects the quality of coverage of that target. However, the proposed algorithms did not prevent collisions between mobile sensors. Hamster is a small, robust and autonomous robot for research and prototype development. The Hamster comes with an onboard Raspberry Pi 3, motor encoder, LIDAR, an IMU and a GPS that enable high precision mapping, localization and path planning algorithms.

~~In this work we have developed a simulation environment in Gazebo framework suited to DCOP_MST problems. This simulation allowed us to test algorithms before execution in the real world, therefore, we could avoid technical issues in early stages of development. We successfully formed a robust net of exchanging messages, both in simulation and in real world conditions.~~

~~In addition,~~ 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, in order to prevent the selection of a single location by more than one agent. We prove for small scenarios that the proposed algorithm converges to the optimal solution, and present empirical evidence that on more complex scenarios, the proposed algorithm maintains high quality coverage, while avoiding collisions. Our empirical study included both software simulation and experiments including a team of sensor carrying robots.

Keywords: ~~Robotic systems, Linux, AI, heuristic, embedded systems, DCOP,~~

Acknowledgements

To Prof. Roie Zivan and Prof. Dan Hermelin, who have helped 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. And special thanks goes to the faculty and to the university, who gave me an opportunity to make this research.

Contents

1	Introduction	1
2	Previous Work & Background	5
2.1	The Distributed Constraint Optimization Problem (DCOP)	5
2.2	DCOP for Mobile Sensing Teams (DCOP_MST)	6
2.3	Standard Max-sum	11
2.4	Adjusting Max-sum algorithm to DCOP_MST	13
2.4.1	Handling Runtime	14
2.4.2	Handling Exploration - FMR	15
2.4.3	Handling Tie Breaking	17
2.5	ROS	17
2.6	Necessity For A New Hardware	19
2.7	Hamster Robots	20
3	Creating Simulations & Setting Robots	21
3.1	Packages	21
3.2	Setting up Gazebo simulation	22

<i>CONTENTS</i>	v
3.3 Example of the full setting	24
3.4 Setting up 2D PyGame Simulation	25
4 Collision Avoiding Max-sum (CAMS)	26
4.1 Analyzing Specific Scenarios	30
4.2 Experimental Evaluation	37
5 Summary & Conclusions	43
6 Future Work	45
6.1 Research Objectives	45
6.1.1 Improving CAMS	45
6.1.2 RL in DCOP and DCOP_MST	46
6.1.3 Dynamic DCOP Problems	47
6.1.4 Competing Groups of Agents	47
6.1.5 Implementation - House IOT	48
6.1.6 Implementation - Allocating Police Units	48
6.2 Expected Advancement	49
7 Declaration	51

List of Figures

2.1	Example with three agents. Dashed outer rings around each agent depict the mobility range. Dark inner rings show the sensing range with the numeric agent credibilities. Stars represent the targets with their coverage requirement. "X"s depict possible locations where the agents can position themselves.	8
2.2	Example of a ROS graph – nodes in the graph represent individual programs; edges represent message streams communicating sensor data, actuator commands, planner states, intermediate representations and so on	18
2.3	Hamsters	20
3.1	Example of different objects existing in simulation	23
3.2	Robots in simulation	23
3.3	(a)	24
3.4	(b)	24
3.5	(c)	24
3.6	(d)	24

3.7	(a) sr and mr of the robots (b) A big field (c) A small field . .	25
4.1	Path $a_4 \rightarrow b_4$ intersects with $c_1 \rightarrow b_1$ and path $b_5 \rightarrow a_5$ intersects with $c_3 \rightarrow b_3$, enabling collisions between robots .	27
4.2	A factor graph generated in CAMS.	29
4.3	Scenarios with convergence guarantees.	30
4.4	target, two robots, one common cell	32
4.5	target, two robots, two common cells	35
4.6	Scenarios with no convergence guarantees.	38
4.7	Remaining coverage (a) and accumulated collisions (b) as a function of the number of iterations.	39
4.8	The set of the experiment.	40
4.9	Time as a function of the number of iterations.	41

List of Tables

1 Introduction

Diverse applications of multi-agent systems have extraordinarily high complexity of a solution, therefore, there is a demand for a distributed optimization approach. Some abstract models and algorithms were developed to represent and solve them. Unfortunately, in numerous real-world cases they are not applicable due to inconsistencies with the theoretical assumptions. Thus, there is a plain need to design some new extensions to those models and algorithms in order to be able to solve such scenarios.

Some of the most laborious multi-agent systems involve teams of mobile sensing agents that are required to acquire information in a given area. Examples include networks of sensors [12, 45] and rescue teams in disaster areas [16]. A crucial, common feature of these applications is that agents select physical locations to move to, and that this selection affects their future interactions, e.g., if a mobile sensor decides to sense a given area, it will then coordinate its actions with nearby sensors.

Such scenarios have been previously modeled using the Distributed Constraint Optimization Problem (DCOP) framework by representing mobile sensors as agents and their tasks/targets as constraints [31]. However, if all possible future moves of dynamic agents are considered, the problem becomes dense. Thus, previous work suggested an iterative process in


which, in each iteration a DCOP instance is built representing the current situation (e.g., sensor positions) and in which only limited movements of the agents are considered. Agents run a distributed algorithm (that might involve several communication cycles) to select the best next joint move, and after they execute it, they build a new DCOP instance considering their new positions [31].

Zivan et al. proposed an extension of the DCOP model and 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 in order to cover a partially unknown environment - *DCOP_MST*. *DCOP_MST* allows agents to adjust their location in order to adapt to dynamically changing environments [45].

The Max-sum algorithm [4, 6, 10] has been the subject of the intensive study in DCOP problem solving research, and has been applied to many realistic applications including sensor networks [9], smart homes [27] and teams of rescue agents [25]. In contrast to standard local search algorithms, agents in Max-sum do not propagate assignments but rather calculate utilities (or costs) for each variable, considering all possible value assignments of their neighboring agents' variables. The general structure of the algorithm is exploitive, i.e., the agents attempt to compute the best costs/utilities for possible value assignments according to their own problem data and recent information they received via messages from their neighbors. A version of Max-sum was also proposed for solving *DCOP_MST* [45]. The main challenge in applying Max-sum to such problems was to overcome the inherent symmetry that the algorithm generates, i.e., either attracting all neighboring sensors to cover a target, which results in a waste, or encouraging all neighboring sensors to explore the

area and not to cover the target. Methods for breaking this symmetry and balancing between exploration and exploitation were proposed in [37].

In this work we address a different limitation of teams of mobile sensing agents, residing on hardware robots: the need to avoid collisions. This requirement was ignored by all previous studies of DCOP_MST. In order to allow the mobile sensors to explore the area, searching for targets and selecting a deployment that maximizes the team coverage, without having collisions of robots, we propose Collision Avoiding Max-sum (CAMS). As in standard Max-sum, the problem in CAMS is represented by a factor graph, which is a bipartite graph including nodes representing variables and functions(constraints), such that nodes from one type only have neighbors of the other type. As in previous attempts to solve problems including dynamic agents with Max-sum, in every iteration (before every movement of the agents), a factor graph is constructed, the algorithm is performed for a limited number of steps¹, and then the agents select the location they move to. The novelty in our work is in the addition of a new type of function-nodes to these factor graphs, which represent locations that agents can choose to move to. This is in contrast to the factor graphs generated by agents in Max-sum_MST that only included function-nodes representing targets [37]. 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: in chapter 2  describe background; within chapters 3 we go through the preparation and an implementation;

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

in chapter 4 we describe the CAMS algorithm and demonstrate gained results; in chapter 5 we summarize this work; at the end, in chapter 6 we discuss the following stages of this research.

2 Previous Work & Background

This chapter presents DCOP, the extended DCOP_MST model, distributed incomplete algorithms and their adjustments to DCOP_MST. In addition, ROS software and Hamster robot hardware are introduced.

2.1 The Distributed Constraint Optimization Problem (DCOP)

Distributed constraint optimization is a general formulation of multi-agent coordination problems that has 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 $\mathcal{A} = \{A_1, A_2, \dots, A_n\}$ is a finite set of agents, $\mathcal{X} = \{X_1, X_2, \dots, X_m\}$ is a finite set of variables, $\mathcal{D} = \{D_1, D_2, \dots, D_m\}$ is the set of finite domains for the variables, and \mathcal{C} is a finite set of constraints¹. Each variable X_i is controlled (or owned) by an agent who chooses a value to assign it from the finite set of values D_i ; each agent may control multiple variables. Each constraint $C \in \mathcal{C}$ is a

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

function $C : D_{i_1} \times D_{i_2} \times \dots \times D_{i_k} \rightarrow \mathbb{R}_+ \cup \{0\}$ 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 (a value assignment to each variable in X). 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 through message passing. 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 there exists at least one constraint that its scope includes a variable controlled by A_i and a variable controlled by A_j . While transmission of messages may be delayed, it is assumed that messages sent from one agent to another are received in the order that they were sent. [17] [43]

2.2 DCOP for Mobile Sensing Teams (DCOP_MST)

The DCOP model makes several assumptions which do not hold in mobile sensor team applications. It assumes that the neighbor set of agents is constant. It also assumes that the constraints, i.e., the utilities/costs incurred by each partial assignment are known a-priori and are constant. Mobile sensors on the other hand are dynamic by nature. The movement of the agents constantly changes the neighbor set. The realistic changing environment results in changing the set of constraints and as a consequence

the utilities for deployment decisions. Consequently, dynamic elements must be formalized and integrated into the DCOP model in order for it to apply to mobile sensor teams (MST).

The DCOP_MST is a dynamic DCOP formulation that models the mobile sensor team coordination problem [45]. The agents $A = \{A_1, A_2, \dots, A_n\}$ in a mobile sensor team are physically situated in the environment, modeled as a metric space with distance function d . The *current position* of agent A_i is denoted by cp_i ; we assume that this position is accurately known by the agent. Locations (or positions) that can be occupied by agents are a finite set of discrete points that form a subset of the total environment. These points can either be a discretization of the underlying space or locations that dominate other nearby points in terms of the sensing quality they afford agents located there. In Figure 2.1, the environment is the Euclidean plane, agents are depicted by small robots, and possible locations are represented by "X"s. Time is discretized so that agents 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 shown in Figure 2.1 by the dashed, outer circle centered on the agent. All "X"s within the circle are locations that the agent can move to in a single time step from its current position.

Agents are only able to effectively sense targets within a limited *sensing range* (sr_i). Because of the sensing range constraint, each agent A_i can observe all targets within a distance sr_i from cp_i , and cannot observe any target that is farther away. The sensing ranges are depicted in Figure 2.1 by the darker, inner circle centered at each agent.

Agents may also differ in the quality of their sensing abilities, a property

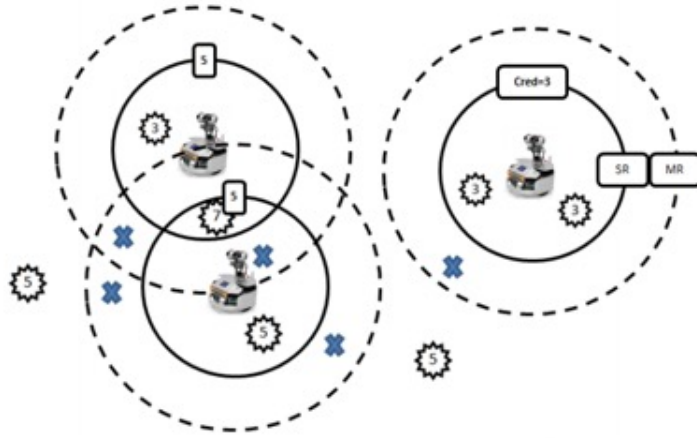


Figure 2.1: Example with three agents. Dashed outer rings around each agent depict the mobility range. Dark inner rings show the sensing range with the numeric agent credibilities. Stars represent the targets with their coverage requirement. “X”s depict possible locations where the agents can position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

termed their credibility. The credibility of agent A_i is denoted by the positive real number $cred(i)$, with higher 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 [35].

The individual credibilities of agents sensing the same target are combined using a joint credibility function $F : 2^A \rightarrow \mathbb{R}$, where 2^A denotes the power set of A . There is a requirement that F has to 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 $\mathcal{T} = \{T_1, T_2, \dots, T_m\}$ 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 (as 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 mobile sensing team problem is to explore the environment sufficiently to be aware of the presence of targets. In the example presented in Figure 2.1 there are seven targets shown as serrated circles and their numbers represent their ER values. Agents within 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

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

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 0:

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

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 results in a lower remaining coverage requirement. In this work: \ominus will be the standard subtraction operator [44].

This constraint C_T for a target T , only involves those agents A_i , whose variable's domain includes a location within the sensing range (sr_i) of T . Therefore, as the domains change, the constraints change as well. As a

consequence, 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 they both move as much as possible in a single time step towards each other. This encodes the fact that such two 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:

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

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 the agents is to position themselves so to minimize:

$$F_{sum}(\mathcal{T}) = \sum_{T_i \in \mathcal{T}} cur_req(T_j)$$

In some cases it may be possible to reduce the values of cur_req 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 aim at minimizing the sum of remaining coverage requirements for all targets. Such a minimization problem is NP-hard [33]. Another possible objective would be to minimize the maximum remaining coverage requirement over all targets. We note that the model and the techniques presented here could be applied to achieve this objective, however, for ease of presentation we do not discuss this here.

2.3 Standard Max-sum

Max-sum [10] operates on a *factor graph*, which is a bipartite graph including nodes that represent variables and constraints [14]. Each variable-node representing a DCOP variable is connected to all function-nodes that represent constraints, which it is involved in. Variable-nodes and function-nodes are considered “agents” in Max-sum, i.e., 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 a 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 a variable-node X to a function-node F in iteration $i \geq 1$ is formalized as follows:

$$Q_{X \rightarrow F}^i = \sum_{F' \in F_X, F' \neq F} R_{F' \rightarrow X}^{i-1} - \alpha$$

Where $Q_{X \rightarrow F}^i$ is the message 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 $R_{F' \rightarrow X}^{i-1}$ 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 $x \in D_X$) in order to prevent the costs carried by messages throughout the algorithm run from growing arbitrarily large.

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

involved in F not including X . The term $\text{cost}(\langle X, x \rangle, PA_{-X})$ represents the cost of a partial assignment $a = \{\langle X, x \rangle, PA_{-X}\}$, which is: $f(a) + \sum_{X' \in X_F, X' \neq X, \langle X', x' \rangle \in a} (Q_{X' \rightarrow F}^{i-1})_{x'}$, 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 $(Q_{X' \rightarrow F}^{i-1})_{x'}$ 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 $\hat{x} \in D_X$ following iteration k as follows:

$$\hat{x} = \arg \min_{x \in D_X} \sum_{F \in F_X} (R_{F \rightarrow X}^k)_x$$

Assuming there are no tied beliefs, Max-sum converges in linear time to the optimal solution when solving problems represented by a tree-structured factor graph [20].² When it 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 [11]. For a more detailed description of the standard Max-sum algorithm please refer to [42]. The Max-sum algorithm has been the subject of intensive study in DCOP problems and has been applied to many realistic applications including mobile sensor networks [7, 8, 22, 26, 30–32] and teams of rescue agents [16, 25].

²Ties can be avoided by adding for each variable-node an unary constraint with extremely small random utilities [10]

2.4 Adjusting Max-sum algorithm to DCOP_MST

The Max-sum algorithm was proposed for solving DCOP_MST in [45] but only in its basic form without any exploration methods or runtime improving techniques. In contrast to standard local search algorithms, agents in Max-sum do not propagate assignments but rather calculate utilities (or costs) for each variable, considering all possible value assignments of their neighboring agents' variables. 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, an increase in the number of agents that can be assigned to tasks would prevent the use of Max-sum for solving such problems. Obviously, extending the local environment of agents results in agents being effective for more tasks/targets and in a larger constraint arity. Therefore, in dynamic scenarios where the time to reach a decision is limited, 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, based on meta-reasoning: agents select for each target a subset of the sensors that can be effective for covering it [38]. The proposed function meta reasoning method (FMR) breaks the relation between the size of the local environment of agents and the arity of the constraints, i.e., the arity of the constraint is not defined by the number of sensors that can be within sensing range of a target t after the next assignment selection (i.e., the "neighbors" of t), but rather by the required number of sensors for covering t . We will discuss FMR further in this section. ~~This version of Max-sum (Max-~~


~~*sum-FMR-Rand*) outperforms the explorative local search algorithms, and other proposed Max-sum approaches to solve DCOP_MST [38].~~

~~Max-sum_MST applied~~ 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 of $mr_i + sr_i$, 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 the process repeats itself.

We further address the challenges that were arisen when applying Max-sum_MST to the DCOP_MST model and how they were solved.

2.4.1 Handling Runtime

While 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 consequently, the factor graph).

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 that are performed before an assignment (position) selection must be selected with care. On one hand, we would like to allow the information regarding the coverage capabilities of sensors to propagate to other sensors. On the other hand, these message cycles of Max-sum result in a single movement for the sensors; thus, we want to avoid unnecessary delays. In experiments of the present research we found that a small number of message cycles (30 in our experimental set-up) was enough to get the best performance.

Regarding the messages of the function-nodes, the only information required to compute the utility by the function is whether a sensor covers the target or not, i.e. there are only two types of positions to execute calculations for. This method reduces the complexity for generating a message by a function-node to $O(D * 2^{K-1})$.

2.4.2 Handling Exploration - FMR

Function Meta Reasoning method takes advantage of a property that is quite common in DCOP_MST, that targets have more neighbors than required for covering them. Consider an iteration i in which the factor graph FG_i was 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 $n(t)_i$. 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 target t 's importance is smaller than $cred_{r(t)_i}$, t can select

$r(t)_i$ neighbors for covering it and allow the other $n(t)_i - r(t)_i$ neighbors to perform exploration. 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 for covering target t , $r(t)$ is a constant number.

Yedidsion et al. propose the following greedy heuristic for selecting the $|r(t)|$ neighbors by a function-node t for which $|n(t)_i| > |r(t)|$. The heuristic is tuned 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 located within sensing range from t and $\bar{n}(t)_i$ includes the rest of the neighbors.
3. While ($|n(t)_i| > |r(t)|$)
 - (1) If ($\bar{n}(t)_i \neq \emptyset$), remove the neighbor in $\bar{n}(t)_i$ that has the highest degree from $n(t)$.
 - (2) Else, remove the neighbor in $\hat{n}(t)_i$ that has the lowest degree from $n(t)$.

It is important to notice 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,

rather it is exponential in $|r(t)_i| - 1$. Thus, the complexity of the computation of function-nodes is no longer dependent on the sensing and mobility ranges of the sensors.

2.4.3 Handling Tie Breaking

The FMR method detaches the connections between targets and agents that are located in a position that allows them to sense the target, yet there is no need for them to do so. The objective of this detachment is to encourage these agents to explore for other targets, where their sensing is required. However, the agents are indifferent between staying in their current location and selecting new locations.

Thus, in order to stimulate exploration, an agent selects an assignment randomly among the tied values that offer the highest utility. This method, denoted as *Rand*, allows the agents to continuously explore new positions and the targets that may be covered from these positions. The objective of tie breaking is simply to coordinate the selection of the same solution among agents. In DCOP_MST on the other hand, agents are incentivized to seek for locations from which they are more effective via the tie breaking method. While it enhances exploration, the *Rand* method has a dichotomous effect in terms of the overall performance of the team.

Onwards, we denote Max-sum_FMR-*Rand* as *Max-sum_MST*.

2.5 ROS

ROS, the Robot Operating System, is an open source framework. ROS is meant to serve as a common software platform for people who are build-

ing and using robots. This common platform lets people share code and ideas more readily. ROS has been remarkably successful. There are over 2,000 software packages, written and maintained by almost 600 people [23]. The main advantage ROS brings to our research is the structure of communication that it provides. It is based on topics and message exchanging between them that allows us to maintain accurately all message flows in the system. Agents can be represented as nodes in the ROS communication system. Each node is able to read (listen) and write (publish) over different net connections in ROS called topics. An example of a wide ROS graph is depicted in Figure 2.2.

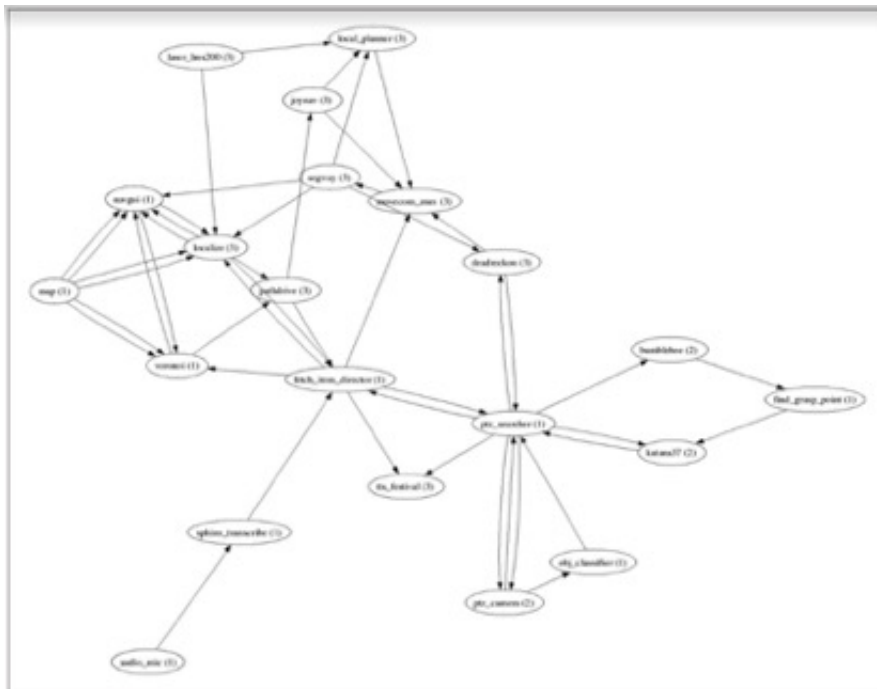


Figure 2.2: Example of a ROS graph – nodes in the graph represent individual programs; edges represent message streams communicating sensor data, actuator commands, planner states, intermediate representations and so on

Another advantage of ROS is the worldwide community of people that provides open source code and sustainable updates. Python is one of the main ROS client libraries, strongly supported by community and relatively simple to use. The community provides a variety of helpful programs we used. For example, Gazebo - creates virtual environments to robots taking into consideration physics laws, Rviz - relieves usage of controllers with graphical interfaces and so on.

The main operating system ROS works with is Linux and more specifically Linux Ubuntu. In our work the version of ROS is "Indigo" and the version of Ubuntu is 16.04.

2.6 Necessity For A New Hardware

In previous implementations of algorithms in DCOP_MST problems carried by Yedidsion, many hardware limitations were raised [36]. Several adjustments were made to the original DCOP_MST model in order to face better with real world physics and obstacles. The new model was named $DCOP_MST^R$ [35]. In $DCOP_MST^R$ robots have their Sensing Range directed only to the front of the robot (unlike in the original model where agents have a 360 degree Sensing Range). This is to cope with the robot's front camera and its limited field of vision. The iRobots used proved to be very difficult to work with. The robots did not have an onboard computer, resulting in a laptop being mounted on every robot. The robots also had no localization capabilities and no communication capabilities. In order to face these issues, it was clear that better hardware was needed that matches specifically this kind of experiments.

2.7 Hamster Robots

Hamster (Figure 2.3) is a small robust autonomous robot for research and prototype development. Hamster can create indoor maps and localize indoors using the 360° LIDAR (LiDar A2M8) and other on-board sensors to provide accurate position information while in motion. It has WiFi ew-7811uac EDIMAX AC600 hardware for a wireless communication. It also has a Camera raspberry Pi module v2 located in front of the robot and it is ROS based. Localization and communication issues are major limitation factor while implementing positioning task with robots. ~~By using the Hamster we can overcome these issues.~~ Its onboard Raspberry Pi 3 also removes the requirement for mounted laptops, making the workflow much easier as now only one computer is needed just for coding and executing. ~~It is then becomes clear that by using the Hamster robots, we can have a much more fruitful experimental environment for testing the DCOP-MST model and its algorithms in the real world.~~



Figure 2.3: Hamsters



3 Creating Simulations & Setting Robots

In this work the words "agent" and "robot" are used interchangeably.

3.1 Packages

It is necessary to understand basic libraries and packages allowing robot control. Here are some usage cases:

1. `actionlib` library enables to send movement commands using `MoveBaseGoal` messages from `move_base_msgs.msg` library. Those messages describe final position of robot in the end of movement.
2. We used `SLAM` algorithm to create a map of an environment. The algorithm is part of built-in packages inside Hamster.
3. We saved those new maps with tools of `map_server` library.
4. To localize itself Hamster uses Augmented Monte-Carlo Localisation algorithm as part of `amcl` package.

5. To create paths in a given environment Hamster builds global and local *costmaps* of surrounding area using `move_base` package. *Costmaps* are special maps used to emphasize obstacles around the robot and thus help to avoid them and to create the shortest and safest path as possible.

We started from very simple tasks like straightforward driving to much more complicated tasks like building maps and creating paths in those maps.

After successful implementation of those tasks we can start to program actual algorithms.

3.2 Setting up Gazebo simulation

Simulation is a safe way to check primary code avoiding technical complications of real robots. In simulation the connection to robot is stable, no battery issues and we do not care about physical damages. That is why it is very convenient to use accurate simulation environment. We used Gazebo framework. In order to create a new world inside Gazebo we need to define descriptions of *objects* (stored in `.stl` and `.dae` files), physical *characteristics* like gravity (stored in `.urdf` files) and *properties* of the world itself describing where each object is located (stored in `.world` file).

As you can see in Figure 3.1 there is a variety of different shapes, materials and mechanisms available in this framework. We use a small fraction of these capabilities: robot-models, walls and simple cylinders.

Fortunately, robot-model in gazebo already provided by "Cogniteam" [5]

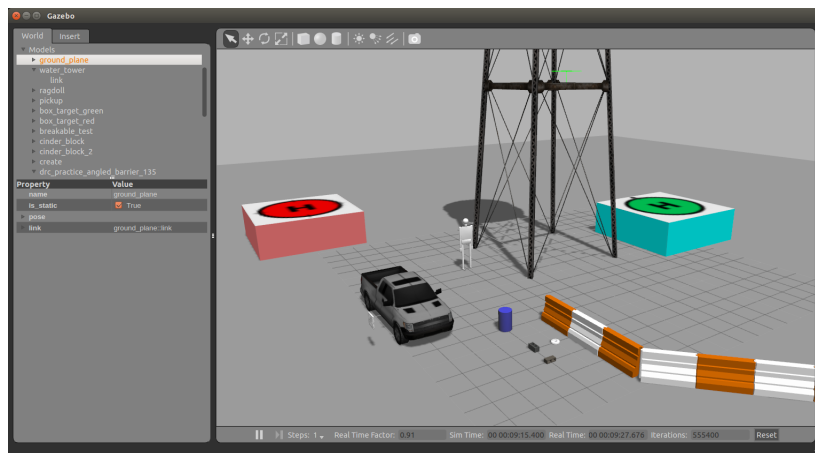


Figure 3.1: Example of different objects existing in simulation

company itself, so we did not need to create it from a scratch. In Figure 3.2 shown the robot-model in a simulation. The model can read and write to ROS topics exactly the same way as it does a real robot. From ROS point of view it does not matter who is standing behind the topics, a simulation or a real robot. In both cases ROS network just provides messages to relevant topics. This property of ROS hugely facilitate development and helps to close gaps between simulation and real conditions.

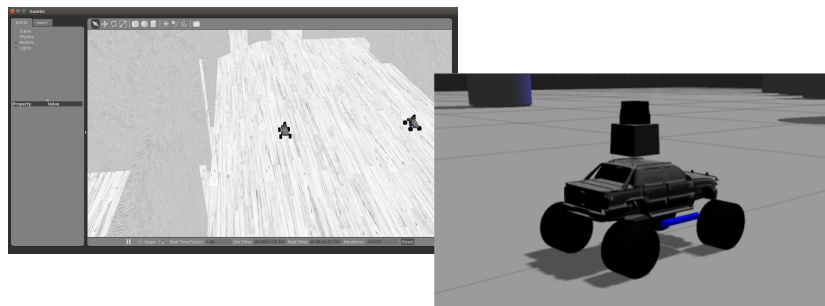


Figure 3.2: Robots in simulation

3.3 Example of the full setting

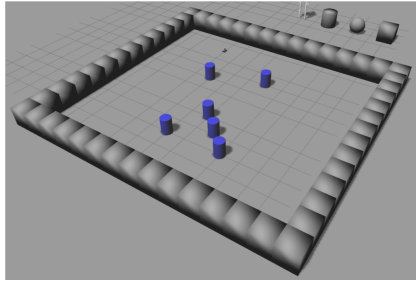


Figure 3.3: (a)

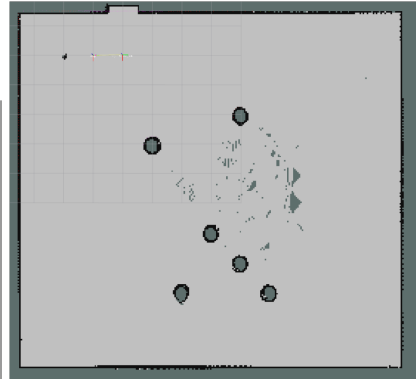


Figure 3.4: (b)

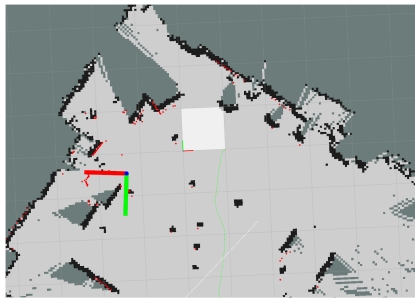


Figure 3.5: (c)

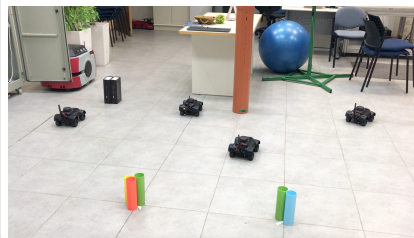


Figure 3.6: (d)

The Gazebo simulation of a simple experiment depicted in Figure 3.3. Blue cylinders are the targets and cubes represent the walls. Figures 3.4 and 3.5 demonstrate maps of the same experiment that were made by `map_server` library in a simulation and in a real world. Figure 3.6 is a snapshot of a real robots setting in a laboratory.

3.4 Setting up 2D PyGame Simulation

In order to test rapidly new ideas and be able to check them on a big scale with dozens of robots and targets on the same field, a simple and light program was needed. We introduce *DCOP_MST 2D simulator*. In figure 3.7 represented some screenshots of the running experiments.

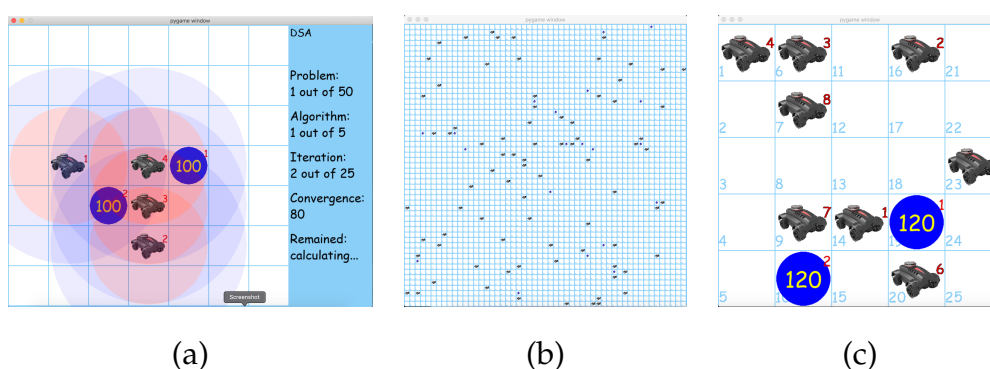


Figure 3.7: (a) sr and mr of the robots (b) A big field (c) A small field

Inside the simulator it is possible to run several algorithms simultaneously. The user can define the size of the field, amount of robots and targets, mr and sr of the robots, number of problems to solve per each algorithm and how many iterations it needed to finish. We can play with requirements of the targets and credibilities of the robots, to set them identical or unique values among all. The ranges, robots' $cred$ values and targets' ER are allowed to change during the run as well. Those capabilities will help US to model better dynamic realistic scenarios in a future research.

As an output the simulator returns: the coverage rate of the algorithms and how many collisions they produced during the run in each iteration.

4 Collision Avoiding Max-sum (CAMS)

According to Yedidsion et al. the Max-sum_MST exploration algorithm greatly improved the solution quality while limiting the exponential runtime associated with message computation in Max-sum. Their empirical study revealed that the benefits of Max-sum_MST enabled it to outperform all competing algorithms including the DSA_MST and MGM_MST versions of the local search [35]. However, this version of Max-sum does not prevent collisions among mobile sensors, which may result in damaging the sensors, execution delay, or even the inability to perform the coverage task. An example of such problems depicted in Figure 4.1. Thus, we propose Collision Avoiding Max-sum (CAMS) that allows the agents to select the deployment that maximizes coverage, while avoiding collisions. This is achieved by adding to the factor graphs that are generated in each iteration of the algorithm (before each movement of the agents), function-nodes representing locations that the agents can move to. Each such function-node can either represent a location to which only one agent can decide to move, or locations to which two agents can move.

The function-node representing the first assigns zero utility for the option

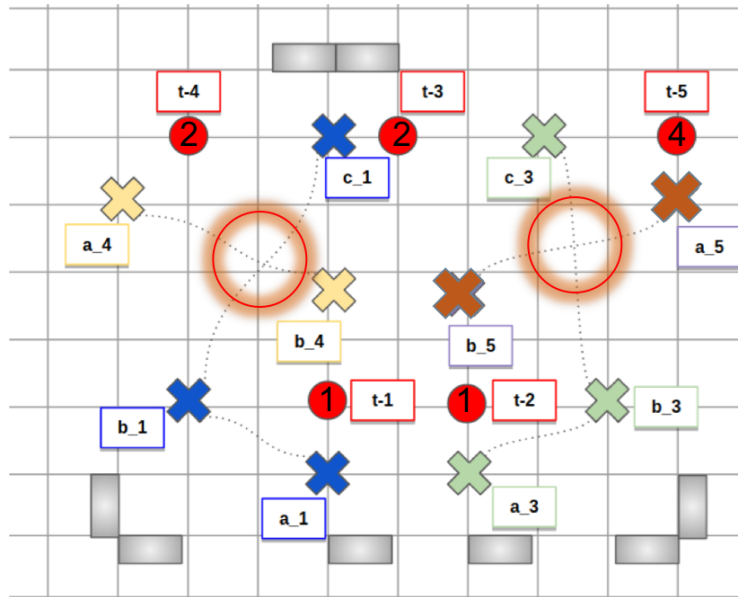


Figure 4.1: Path $a_4 \rightarrow b_4$ intersects with $c_1 \rightarrow b_1$ and path $b_5 \rightarrow a_5$ intersects with $c_3 \rightarrow b_3$, enabling collisions between robots

that the agent does not choose to move to the location it represents and a positive utility for the option that it does.

The second type represents a hard constraint that prevents two agents from selecting the same location. In this case, its utility function includes zero utility for the option that both agents do not move to the represented location, positive utilities for the combinations in which only one agent selects to move to this location, and minus infinity for the option that the two agents move to this location.

In case more than two agents can select a location, the agents generate function-nodes that represent the binary constraint between each pair among them.

More formally, in a factor graph generated by agents in CAMS there are three types of function-nodes:

1. FT_j , a function-node representing a target T_j . As in Max-sum_MST, the utility an agent A_i derives for covering T_j is $\min\{ER_{FT_j}, cred_i\}$. However, in non-degenerate cases where multiple sensors are required for coverage, the utility derived is $cred_i$.
2. $FL_{(i,e)}$, a function-node representing a location l to which agents A_i and A_e can move in this iteration. The utility for both agents not selecting location l is zero, for both agents selecting l is $-\infty$ and for both options in which only one of them selects l , a random utility is selected from a range of numbers that is much smaller than ER_{FT_j} ¹. We emphasize that, although there may exist scenarios in which more than two agents can move to the same location, FL is defined as a binary constraint, and thus, if there are $k > 2$ agents that can select the same location, there will be an FL for each pair of these k agents.
3. $FL_{(i)}$, a function-node representing a location to which only one agent can move to. In this case the corresponding constraint is unary. A random positive number is selected for the option that A_i selects this location (selected from the same range as the random positive utilities selected for the binary constraints) and zero for not selecting this location.

Figure 4.2 presents an example of a factor graph generated in some iteration of CAMS. It includes two mobile sensors, each with four possible locations to move to (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 the third type. While the domain of

¹Random numbers are selected to avoid ties between desired options, as in [10]

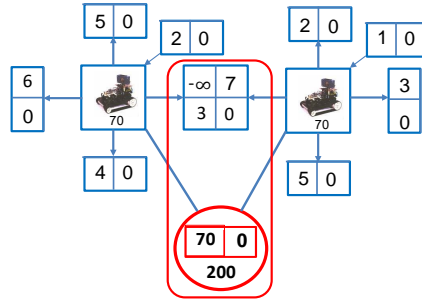


Figure 4.2: A factor graph generated in CAMS.

each agent includes five values (representing the possible locations it can select), only for the selection of the location represented by the function-node the utility is positive and for all other locations it is zero. The middle location to which both mobile sensors can move, is represented by a function-node of the second type. It includes four options, one for both agents not selecting this location (zero utility), one for both agents selecting this location (minus infinity) and two with positive utilities for the cases that only one agent selects this location. The target is represented by a function-node of the first type. Its coverage requirement is 200, while the credibility of each mobile sensor is 70, which is the utility they derive for covering the target. In this example, covering the target is only possible from the middle location that both mobile sensors can move to. However, if they both move to this location they collide.

The overhead run-time complexity of CAMS (Compared to Max-sum_MST) is negligible, since the additional function-nodes representing unary and binary constraints require at most 2^2 utility comparisons for each message produced. On the other hand, the comparisons required for generating a message by each target representing function-node FT (in both algorithms) is 2^k , where k is number of neighbors of FT . However, one may expect that the addition of location representing function-nodes will

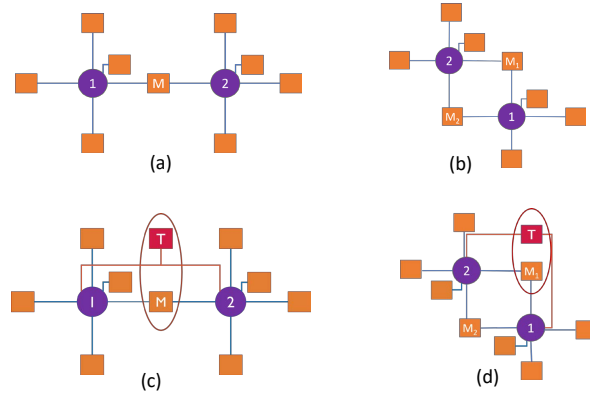


Figure 4.3: Scenarios with convergence guarantees.

result in more cycles and consequentially, reduce the probability for convergence. We demonstrate in the following sections that this is not the case in CAMS.

4.1 Analyzing Specific Scenarios

Next we present a set of scenarios for which we prove the convergence of CAMS to a non colliding (optimal) state. A second set including scenarios for which our empirical results demonstrated such convergence, will be presented in the following section.

The scenarios for which we established convergence guarantees are depicted in Figure 4.3. ~~Sketch proofs for these cases~~ are listed next.

Proposition 1 *Max-sum converges to a collision free optimal solution, in a linear number of steps, when solving the scenario depicted in Figure 4.3 (a).*

Proof: The factor graph representation of this scenario has a tree structure and thus, the algorithm will converge in a linear number of steps to an

optimal solution [41]. This optimal solution cannot include the selection of the mutual location M by both agents, since its cost is $-\infty$. \square

Proposition 2 *Max-sum converges to a collision free optimal solution in a pseudo linear number of steps, when solving the scenario depicted in Figure 4.3 (b).*

Proof: The factor graph representation of this scenario includes a single cycle with two function-nodes. Each of these function-nodes has four entries in their utility table. According to [11], when belief propagation is applied to a single cycle graph, it 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 two function-node four entry utility table cycle is when it 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 is dependent on the constant utilities sent by the unary function-node neighbors, which are not included in the cycle. If the difference between these utilities is negligible, the time for convergence is linear, i.e., in the order of the size of the cycle. \square

The following Lemma will be relevant to scenarios (as depicted in Figure 4.2), which their representing factor graph includes a single target function-node and a single location from which the target can be covered and both agents can move to.

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

²A Previous indication that hard constraints reduce the complexity of a factor graph

Proof: We will follow the path of messages in 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 will 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 . We recall that by construction, $cred_1$ and $cred_2$ are much larger than c_1 and c_2 . A factor graph that applies to this description is depicted in Figure 4.4.

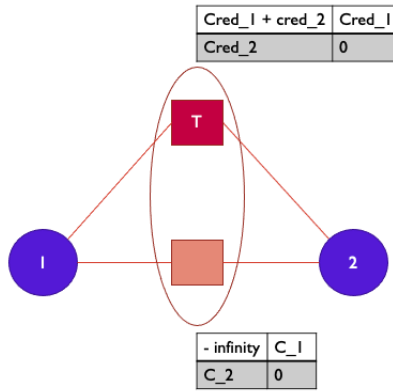


Figure 4.4: target, two robots, one common cell

Without loss of generality, we will follow the cyclic message path starting at FT , then MS_1 , ML , MS_2 and 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 beliefs for 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 to pass further.

representation of a realistic application (although very different from the case this Lemma analyzes) can be found in [21]

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 $\langle c_2, cred_1 + c_1 \rangle$.

4. The fourth message $MS_2 \rightarrow FT$ passes forward the same $\langle c_2, cred_1 + c_1 \rangle$.
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, whether $cred_2 + c_2 > cred_1 + c_1$ or not, the message ~~we~~ be normalized down to $\langle 0, cred_1 \rangle$.

Now we will follow the cyclic message in the opposite order of nodes starting at FT , then MS_2 , ML , MS_1 and 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 for 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 to pass further.
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 forward the same $\langle c_1, cred_2 + c_2 \rangle$.

5. The next $FL \rightarrow MS_2$ message 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, whether $cred_2 + c_2 > cred_1 + c_1$ or smaller, the message ~~we~~ be normalized down to $\langle 0, cred_2 \rangle$.

We emphasize that a similar analysis applies for any case where $cred_1$ and $cred_2$ are much larger than c_1 and c_2 . \square

Intuitively, this happens because the $-\infty$ utility in ML eliminates 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 has shifted from the covering option to the non-covering option and this offsets the utility added by FT in each cycle.

Proposition 3 *Max-sum converges to a collision free optimal solution, when solving the scenario depicted in Figure 4.3 (c), after a linear number of steps.*

Proof: According to Lemma 1 the target representing 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 is solving a tree. \square

An immediate corollary is that Max-sum will converge in scenarios similar to the scenario depicted in Figure 4.3 (c), in which the target can be covered from additional locations, since the graph will still have a single degenerate cycle, i.e., the algorithm will perform as if it was solving a tree-structured factor graph.

For analyzing the convergence of the scenario represented by the factor graph depicted in Figure 4.3 (d) we state the following Lemma:

Lemma 2 When Max-sum operates on factor graphs as depicted in Figure 4.3 (d), in every step, the target function-node FT will send the same messages to its two neighbors MS_1 and MS_2 , 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 the proof of Lemma 1. We will 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 location ML_1 and ML_2 to which both sensors can move. We will 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 or only MS_2 move to ML_1 or ML_2 respectively. We recall that by construction, $cred_1$ and $cred_2$ are much larger than c_1, c_2, c_3 and c_4 . A factor graph that applies to this description is depicted in Figure 4.5.

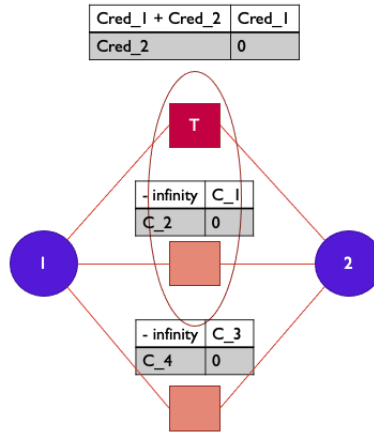


Figure 4.5: target, two robots, two common cells

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

1. The first step:

$FT \rightarrow MS_1$ 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. The second step:

$MS_1 \rightarrow ML_1$ 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. The third step:

$ML_1 \rightarrow MS_2$ 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. The fourth step:

$MS_2 \rightarrow FT$ includes: $\langle c_2, cred_1 + c_1 + c_3 \rangle$

We will denote c_2 as α and $c_1 + c_3$ as β .

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

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

In both cases, whether $cred_2 + \alpha$ is bigger than $cred_1 + \beta$ or smaller, the message we be normalized down to $\langle cred_1, 0 \rangle$.

We emphasize that a similar analysis applies for the opposite order of nodes, where $cred_1$ and $cred_2$ are much larger than c_1, c_2, c_3 and c_4 . \square

Proposition 4 *Max-sum converges to a collision free optimal solution, in a pseudo-linear number of steps, when solving the scenario depicted in Figure 4.3 (d).*

Proof: Similar to the proof of Proposition 3. According to Lemma 2 the target representing 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 is solving a single cycle factor graph. This cycle cannot include an inconsistent optimal path (same argument as stated for Proposition 2) and thus, the algorithm will converge to the optimal solution, which cannot include collisions. We omit the similar argument for the pseudo-linear number of steps required for convergence. \square

4.2 Experimental Evaluation

In order to evaluate the performance of CAMS we designed two types of simulation environments. The first was software simulation, implemented in Python (Chapter 3) and the second was a simulation that included Hamster robots [5]. We started by evaluating the performance of Max-sum on small scenarios including at most two targets, on which we were not able to establish guaranteed convergence (depicted in Figure 4.6).

For each scenario we produced 50 instances. In all of them the target's ER values were 120 and the credibility of each sensor was 30. The positive utilities of location function-nodes were selected randomly between 1 and $1M$, and divided by $10,000M$ (resulting in random numbers in the range $[0.00000000001, 0.0001)$). Our empirical evaluation revealed that Max-sum always converged 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 the standard deviation (on the right) are depicted in brackets.

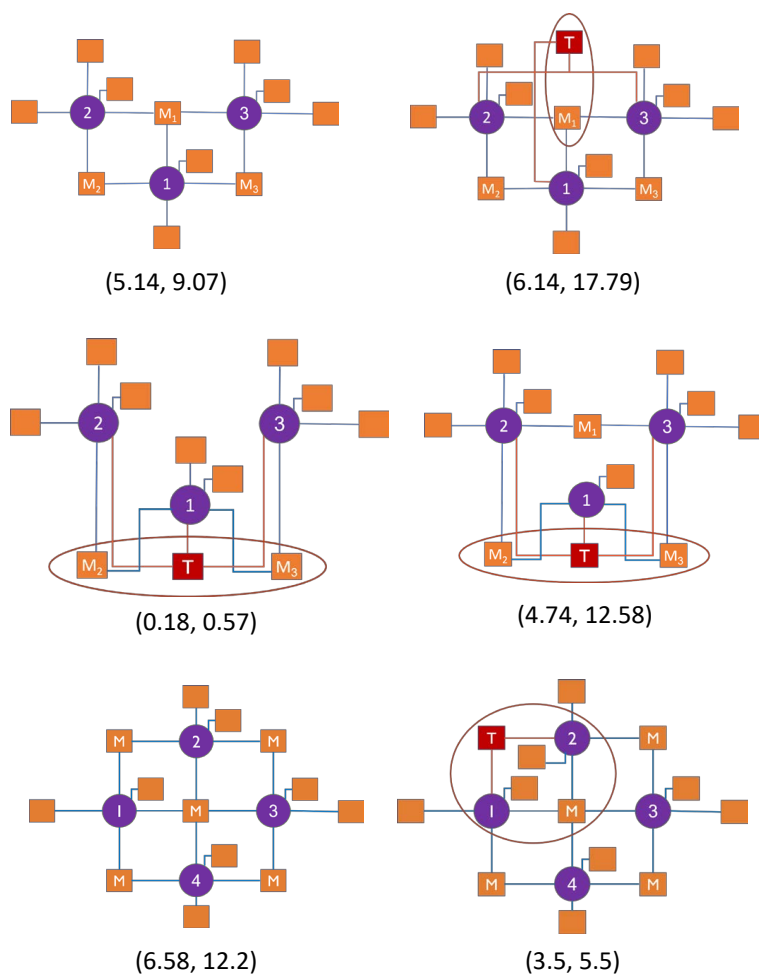


Figure 4.6: Scenarios with no convergence guarantees.

Next, we generated larger scenarios in which 20 targets were randomly positioned on a grid with dimensions 50×50 . 80 mobile sensors were also positioned randomly on this grid, such that every cell of the grid included at most one mobile sensor. The targets' ER values, the credibility of mobile sensors and the utilities for the cell representing function-nodes, were selected as for the small scenarios described above. In each iteration of the algorithm, each mobile sensor could 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 that is located in the cells closest to its location in each direction, including the row, the column and the diagonals (one step in each direction).

We compared CAMS with Max-sum_MST and a random walk algorithm. Each algorithm performed 100 iterations, in which the mobile sensors selected locations. CAMS and Max-sum_MST performed 30 steps of the algorithm in each iteration, before the agents selected their locations. The remaining coverage in each iteration was calculated as follows $\sum_{T_j \in T} cr(T_j)$ (thus, the group goal was to minimize the remaining coverage requirement).

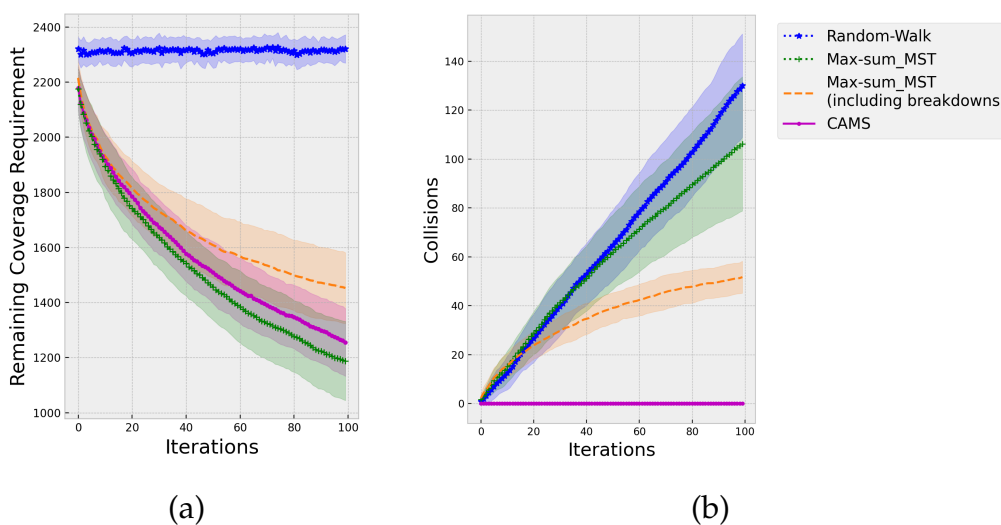


Figure 4.7: Remaining coverage (a) and accumulated collisions (b) as a function of the number of iterations.

Figure 4.7 (a) presents the remaining coverage requirement for the three algorithms as a function of the number of iterations (colors represent standard deviation). For Max-sum_MST we included the results of two experiments. In the first, the mobile sensor movements were not affected

by collisions. In the second, colliding sensors exhibited a breakdown and stopped moving. It is clear that CAMS maintains a similar level of coverage to Max-sum_MST and both have a large advantage over the random walk algorithm. However, since the mobile sensors in Max-sum_MST do not avoid collisions, the agents performing it are less restricted and therefore the resulting coverage is slightly better than the coverage achieved when performing CAMS (differences were found to be insignificant with $p = 0.01$). On the other hand, CAMS significantly outperformed Max-sum_MST in the scenarios including breakdowns in terms of coverage. Figure 4.7 (b) presents the number of accumulated collisions of the algorithms. CAMS is collision free while random walk and Max-sum_MST perform collisions in a similar rate.

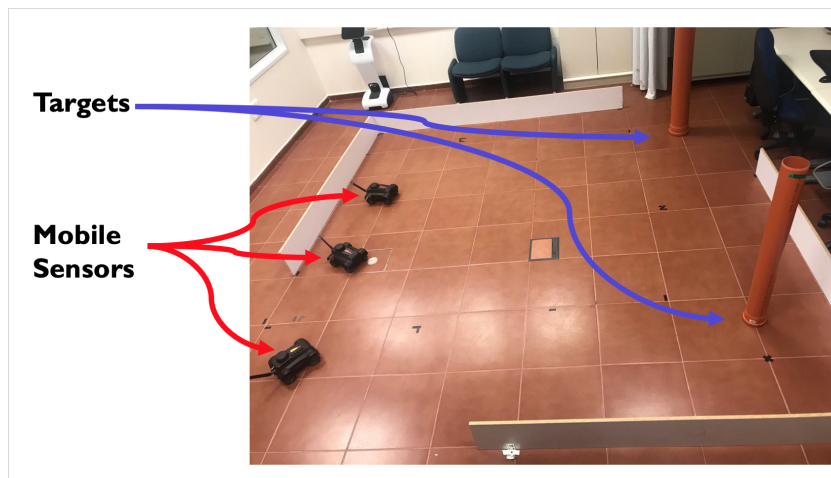


Figure 4.8: The set of the experiment.

In the next set of experiments, our goal was to examine the delay caused by collisions between robots in realistic settings. Thus, they included a mobile sensing team composed of three Hamster robots [5] and two targets, placed in a 4×4 grid, where the size of each cell was a square meter. The targets' ER was set to 60. They were placed randomly in non adja-

cent cells of the grid. The sensors' credibility and the number of steps the Max-sum algorithm was performed in each iteration were identical to all experiments presented in this section. The number of iterations the algorithms performed was 10. We selected four different positions for the targets and for each of them, five different positions for the robots, resulting in 20 experiments for each algorithm.³ The start position of one of the experiments is depicted in Figure 4.8.

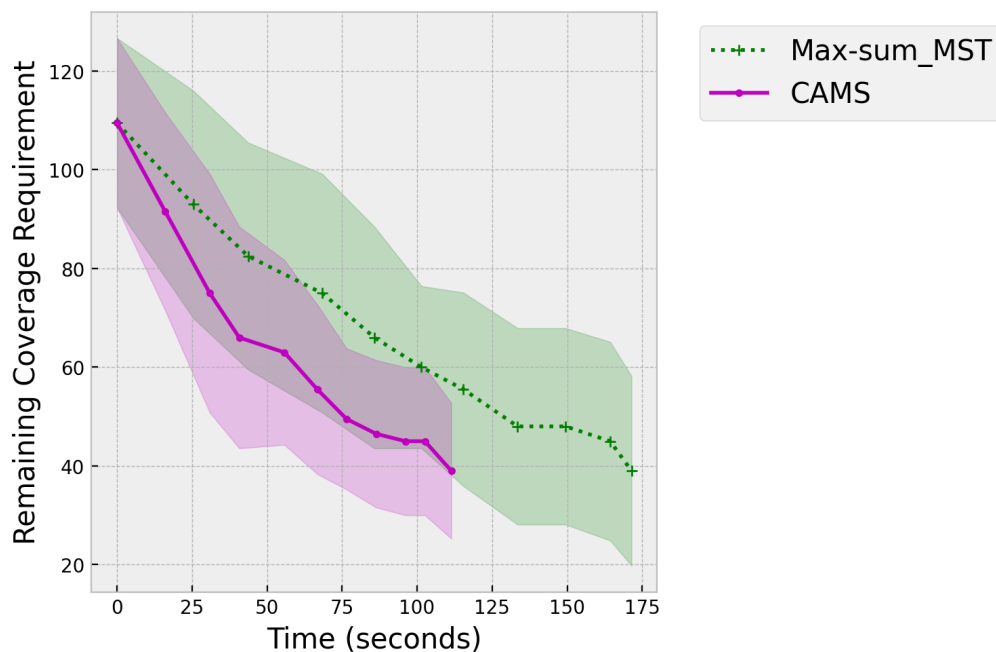


Figure 4.9: Time as a function of the number of iterations.

Figure 4.9 presents the remaining coverage requirement as a function of the experiment execution time. Both algorithms produce the same level of coverage after completing ten iterations. However, CAMS avoids colli-

³A video presenting the highlights of this set of experiments is included in the supplementary material.

sions, and thus, reaches this coverage state faster.

5 ~~Summary & Conclusions~~

~~Mobile Sensor Team (MST) is a scenario where a team of robots has to detect targets and keep a track on them. DCOP_MST is a dynamic DCOP formulation. Using the DCOP_MST model, we model the MST scenario and use various algorithms to solve it. The DCOP_MST has been tested in a real world environment through a team of iRobot robots. The robots had many hardware limitations which made it very difficult to produce high quality results.~~

~~In this work we took an attempt at testing DCOP_MST in a real world environment with a team of a new type of robots, called Hamster. The Hamster robots feature better localization and communication capabilities which makes them much more suitable for this research.~~

~~Operation and implementation of the DCOP_MST algorithms into the Hamster robots proved to be a challenging task, as the initial configuration of these robots was very basic. We created several useful instruments, such as: simulations, mapping tools, robust network for exchanging messages between robots, navigation and localization tools.~~

An important feature of applications that include mobile sensors, is that they should avoid collisions, while optimizing coverage. In order to achieve this challenging combination, we proposed CAMS, a version of Max-sum_MST

that avoids collisions by adding to the factor graph representation of the problem, besides target representing function-nodes, hard constraint function-nodes, representing the locations that agents may choose to move to. In contrast to what one might expect, this addition did not prevent the algorithm from converging. Our analysis of simple scenarios proposed explanations for this phenomenon and our empirical results revealed that the desired properties are maintained when the problem scales. Small scenario experiments with hardware robots revealed the actual delay in execution caused by collisions.

6 Future Work

~~Our work prepared platform for future research and development in a multi-agent world, more specifically in DCOP and DCOP-MST.~~ We outline several options for future research. It will include theoretical and practical algorithmic analysis and design, which takes into consideration applications' peculiarities. We intend to investigate whether existing algorithms maintain their properties in ~~those different~~ environments that include message latency and ~~losses~~, changes in search space, diverse behavioral characteristics of agents, ~~etc.~~ In addition, the design will address the robustness of the algorithms with respect to existing approaches.

6.1 Research Objectives

6.1.1 Improving CAMS

Some additional exploration techniques of exploration were proposed in [35] in order to improve performance of the algorithms. We are willing to apply those techniques and run real robot experiments in a greater scale.

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 [2, 15, 24, 28, 34]. 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 target detection.

6.1.2 RL in DCOP and DCOP_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. [39]

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[18]. (ii) Introduction of distributed reinforcement

learning algorithms, that balance exploration and exploitation to solve MD-DCOPs in an online manner. [18, 40] (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. [39]

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.

6.1.3 Dynamic DCOP Problems

In this work we made an assumption that target's demand and position are static during the run of algorithms. However, many real world scenarios have highly dynamic nature [16, 19, 29, 36]. Moreover, Our DCOP_MST model works in 2D plane. Drones, however, position themselves in a 3D space, which makes a search space at list an order of magnitude more complex [1]. The targets can move in space, they can appear in different times and their demand can change dramatically as well. We are willing to explore the performance of existing DCOP_MST algorithms in such challenging environments. For instance, we can use previous iRobot robots as mobile targets, we can use drones as agents.

6.1.4 Competing Groups of Agents

Another assumption we took in this work is that a group of agents works as a cooperative team. In many real world examples there are competing

teams of agents trying to solve similar task, such as parlor games, competitive economic situations, and some social choice settings [3]. It leads to game-theoretic approach in DCOP. We are willing to model those scenarios in DCOP formulations and transform existing algorithms to solve those kind of problems.

6.1.5 Implementation - House IOT

Recently, a realistic dataset for the smart home device scheduling problem for DCOPs was published [13], and several attempts to solve this problem were taken [27]. Smart house problem creates supplementary challenges such as, more sophisticated factor-graph, great number of variables and agents (each smart property in house can be associated to a separate agent), creating a proper DCOP formulation to this problem, etc. We are aiming to adapt new algorithms such as new versions of Max-sum to this implementation.

6.1.6 Implementation - Allocating Police Units

Another DCOP implementation was proposed in [25]. They looked at decentralized coordination in RoboCup rescue challenge. Emergency agents are faced with a number of significant challenges while managing major disasters. First, the number of rescue tasks posed is usually larger than the number of agents and the resources available to them. Second, each task is likely to require a different level of effort in order to be completed by time. Third, new tasks may continually appear or disappear from the environment, thus requiring the agents to quickly recompute their allocation of resources. Fourth, forming teams or coalitions of multiple agents

from different agencies is vital since no single agency will have all the resources needed to save victims, unblock roads, and extinguish the fires which might erupt in the disaster space.

In their results they came with some open questions such as, how the proposed algorithms scale with increasing numbers of agents and different profiles of deadlines and workloads, how to evaluate the performance of those algorithms under more general settings of variable and factor degrees and how to extend the algorithms to consider the addition and removal of agents from the environment and to achieve convergence on cyclic graphs. We see that our knowledge in solving cyclic graphs can help to approach this problem.

6.2 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.

7 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 list of references.

Bibliography

- [1] F. Al-Turjman. A novel approach for drones positioning in mission critical applications. *Transactions on Emerging Telecommunications Technologies*, n/a(n/a):e3603. e3603 ETT-18-0435.R1.
- [2] Z. Bnaya, R. Stern, A. Felner, R. Zivan, and S. Okamoto. Multi-agent path finding for self interested agents. 01 2013.
- [3] F. Brandt, F. Fischer, P. Harrenstein, and Y. Shoham. A game-theoretic analysis of strictly competitive multiagent scenarios. In *IJCAI 07*, Hyderabad, India, January 2007.
- [4] 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.
- [5] Cogniteam. *Hamster V7 Smart ROS Autonomous Ground Vehicles for Industry and Academic R&D*, 2020.
- [6] 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.

- [7] Farinelli, Rogers, Petcu, and Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS*, 2008.
- [8] A. Farinelli, A. Rogers, and N. Jennings. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *JAA-MAS*, 2013.
- [9] A. Farinelli, A. Rogers, and N. R. Jennings. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Journal of Autonomous Agents and Multi-Agent Systems (JAA-MAS)*, 28(3):337–380, 2014.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] W. Kluegel, M. A. Iqbal, F. Fioretto, W. Yeoh, and E. Pontelli. A realistic dataset for the smart home device scheduling problem for dcops. *CoRR*, abs/1702.06970, 2017.

- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [21] T. Peña-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.
- [22] 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.
- [23] M. Quigley, B. Gerkey, and W. D. Smart. *Programming Robots with ROS*. O’Reilly Media, 2015.
- [24] V. Rahmani and N. Pelechano. Multi-agent parallel hierarchical path finding in navigation meshes (ma-hna*). *Computers & Graphics*, 86, 11 2019.
- [25] S. D. Ramchurn, A. Farinelli, K. S. Macarthur, and N. R. Jennings. Decentralized coordination in robocup rescue. *Computer Journal*, 53(9):1447–1461, 2010.
- [26] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 2011.
- [27] 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.
- [28] 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.

- [29] 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.
- [30] R. Stranders, F. M. Delle-Fave, A. Rogers, and N. R. Jennings. A decentralised coordination algorithm for mobile sensors. In *AAAI*, 2010.
- [31] 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.
- [32] 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.
- [33] 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.
- [34] K.-H. C. Wang and A. Botea. Tractable multi-agent path planning on grid maps. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI'09*, page 1870–1875, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.
- [35] H. Yedidsion. *Distributed Constraint Optimization for Teams of Mobile Agents*. PhD thesis, Ben Gurion University, 2015.
- [36] H. Yedidsion and R. Zivan. Applying dcop_mst to a team of mobile robots with directional sensing abilities (extended abstract). In *AAMAS*, 2016.

- [37] 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.
- [38] H. Yedidsion, R. Zivan, and A. Farinelli. Applying max_sum to teams of mobile sensing agents. In *EAAI*, 2018.
- [39] C. Zhang and V. Lesser. Coordinating multi-agent reinforcement learning with limited communication. volume 2, pages 1101–1108, 05 2013.
- [40] 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.
- [41] 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.
- [42] 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.
- [43] 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.
- [44] R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, and K. Sycara. Distributed constraint optimization for teams of mobile sensing agents. In *JAAMAS*, 2015.

- [45] R. Zivan, H. Yedidsion, S. Okamoto, R. Grinton, 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.