Journal of Artificial Intelligence Research 79 (2024) 1281-1311 Submitted 11/2023; published 04/2024

# Collision Avoiding Max-Sum for Mobile Sensor Teams

**Arseni Pertzovskiy** ARSENIP@POST.BGU.AC.IL
**Roie Zivan** ZIVANR@BGU.AC.IL
*Department of Industrial Engineering and Management*
*Ben-Gurion University of the Negev*
*Beer Sheva, Israel*

**Noa Agmon** AGMON@CS.BIU.AC.IL
*Department of Computer Science and Engineering*
*Bar-Ilan University*
*Ramat Gan, Israel*

## Abstract

Recent advances in technology have large teams of robots with limited computation skills working together to achieve a common goal. Their personal actions need to contribute to the joint effort, however, they also must ensure that they do not harm the efforts of the other members of the team, e.g., as a result of collisions. We focus on the distributed target coverage problem, in which the team must cooperate to maximize utility from sensed targets while avoiding collisions with other agents. State-of-the-art solutions focus on the distributed optimization of the coverage task at the team level while neglecting to consider collision avoidance, which could have far-reaching consequences on the overall performance. Therefore, we propose CAMS: a collision-avoiding version of the Max-sum algorithm, for solving problems including mobile sensors. In CAMS, a factor-graph that includes two types of constraints (represented by function-nodes) is iteratively generated and solved. The first type represents the task-related requirements, and the second represents collision avoidance constraints. We prove that consistent beliefs are sent by target representing function-nodes during the run of the algorithm, and identify factor-graph structures on which CAMS is guaranteed to converge to an optimal (collision-free) solution. We present an experimental evaluation in extensive simulations, showing that CAMS produces high-quality collision-free coverage also in large and complex scenarios. We further present evidence from experiments in a real multi-robot system that CAMS outperforms the state of the art in terms of convergence time.

## 1. Introduction

Some of the most challenging multi-agent applications involve teams of mobile sensing agents that are required to acquire information in a given area. Examples of such applications are networks of sensors (Jain, Taylor, Yokoo, & Tambe, 2009; Zivan, Yedidsion, Okamoto, Glinton, & Sycara, 2015), smart homes (Rust, Picard, & Ramparany, 2016), and rescue teams in disaster areas (Macarthur, Stranders, Ramchurn, & Jennings, 2011). The mobile agents reside on physical aerial or ground devices, thus, they must avoid collisions. Moreover, the dynamic nature of the environments the agents operate in, as well as their commonly limited computation and sensing capabilities, require that the decisions made by each agent are fast and short-term. The ability of those local short-term decisions to result in an overall optimal group strategy is a fundamental challenge for multi-agent systems, and it is the focus of this paper. Specifically, we examine the problem of *distributed target coverage* by a team of limited robots, in which the robots decide which targets to sense for yielding maximal group utility.

Distributed constraint optimization problems (DCOP) offer a framework that addresses some of the above challenges. As DCOPs are limited in representing dynamic events, an extension of the DCOP framework, DCOP_MST (Mobile Sensor Team), along with local search algorithms, were proposed by Zivan et al. (2015). A later study has shown that an incomplete inference algorithm, Max-sum (Farinelli, Rogers, Petcu, & Jennings, 2008; Chen, Deng, Wu, & He, 2018; Deng & An, 2020), produces better results when used in an iterative process for solving DCOP_MST, where iterative instances of the current representation of the problem are solved distributively and allow the agents to select the next joint move (Yedidsion, Zivan, & Farinelli, 2018). The Max-sum algorithm has been the subject of intensive study in DCOP solving research and has been applied to many realistic applications (Farinelli, Rogers, & Jennings, 2014; Rust et al., 2016; Ramchurn, Farinelli, Macarthur, & Jennings, 2010), among those for solving DCOP_MST, by the Max-sum_MST algorithm (Zivan et al., 2015). Previous studies that investigated the performance of Max-sum when solving standard DCOPs (e.g., (Zivan, Parash, Cohen, Peled, & Okamoto, 2017; Cohen, Galiki, & Zivan, 2020)) reported that Max-sum (without the addition of function-node splitting) oscillates for thousands of iterations, whether it finally converges or not. The nature of this phenomenon has remained an open question. In contrast, when applied to DCOP_MST, Max-sum converges instantly (Yedidsion et al., 2018).

While previous DCOP-based work offers solutions that can be successfully applied to target coverage, they neglect to consider in their optimization criteria one critical aspect of the problem: collision avoidance. Since the team members act physically in the environment, they cannot collide with each other. As seen vastly in robotics research, and in the research area of Multi-Agent Path Finding (MAPF), accounting for collisions between physical agents is extremely challenging, and has far-reaching consequences on the performance of the system (Stern, Sturtevant, Felner, Koenig, Ma, Walker, Li, Atzmon, Cohen, Kumar, et al., 2019; Hoy, Matveev, & Savkin, 2015). However, as opposed to MAPF where the goal is to globally create collision-free paths optimizing some joint path-length criterion, or collision-avoiding in robotics research that focuses on generating locally safe trajectories, here we are interested in *local* decision-making for *target-coverage optimization*, where collision-avoidance being an additional important constraint.

Therefore this paper addresses the two above important challenges that arise when using incomplete distributed inference algorithms for solving dynamic mobile sensing team problems. The first is the enigma related to the fast convergence of Max-sum when applied to DCOP_MST. We prove that the convergence results from the special structure of MST problems and that the message content sent by the nodes representing the constraints in Max-sum will be fixed when solving DCOP_MST.

The second challenge that we address is the need to avoid collisions while solving local optimization problems, modeled as DCOPs. We propose a collision-avoiding version of Max-sum, called 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). In addition to the standard optimization using Max-sum, CAMS adds a new type of function-nodes to the factor-graphs, which represent the constraints of locations that agents can choose to move to.

We identify structures of factor-graphs on which CAMS is guaranteed to converge to the optimal (collision-avoiding) solution. Moreover, we prove that on any graph structure, if the algorithm converges, it is to a collision-free solution. We also prove that the added complexity of CAMS with respect to Max-sum_MST is small. We show empirically in extensive simulations and using real robotic systems that even for complex cases (e.g., in dense and dynamic environments), CAMS converges efficiently to high-quality collision-free solutions.

This paper is an extension of our AAMAS 2023 paper (Pertzovskiy, Zivan, & Agmon, 2023). Besides the extended discussion, this version of the paper includes a formal definition of the collision avoidance DCOP_MST problem (CA-DCOP_MST), a proof that whenever our proposed algorithm CAMS converges, it is to a collision-free solution, and an extended empirical evaluation that includes diverse environments, comparison between versions of the algorithm that use different utility adaptation methods (i.e., ordered value propagation (OVP) and balanced utility adaptation (BUA)) and an extended description of our empirical evaluation with a robot team.

## 2. Related Work

Handling collisions by a team of robots in the target-coverage task using DCOPs relates to various research directions in the literature, all of which are presented in this section.

The DCOP model has been widely used for representing and solving coordination problems related to sensor networks (Farinelli et al., 2014; Nguyen, Yeoh, Lau, Zilberstein, & Zhang, 2014) and mobile sensor networks (Stranders, Farinelli, Rogers, & Jennings, 2009; Taylor, Jain, Jin, Yokoo, & Tambe, 2010; Zivan et al., 2015). To the best of our knowledge, *none of these studies addressed the possibility of collisions between mobile sensors*. Nevertheless, an attempt to use the DCOP model and algorithms in order to avoid collisions of ships was presented in (Hirayama, Miyake, Shiota, & Okimoto, 2019), proposing the distributed stochastic search algorithm (DSSA) for preventing ships from selecting colliding routes. We compare this version and our proposed algorithm with a collision-avoiding DSA (CADSA) algorithm in our empirical study.

DCOPs are traditionally associated with problems in discrete settings (Stranders et al., 2009; Taylor et al., 2010; Zivan et al., 2015; Zivan, Lev, & Galiki, 2020), including DCOP_MST. Attempts to investigate the modeling and solving of DCOPs in continuous domains (e.g., (Voice, Stranders, Rogers, & Jennings, 2010; Hoang, Yeoh, Yokoo, & Rabinovich, 2020; Sarker, Choudhury, & Khan, 2021)), raise challenges concerning, among others, the type of continuous utility/cost functions and the impact of these types on the complexity of the problem. While the importance of generalizing DCOPs to continuous domains is noted, this paper follows the common discrete modeling, associated also with DCOP_MSTs (which is the baseline for this work).

Different aspects of the Max-sum algorithm have been examined in the literature, focusing on the algorithm's convergence guarantees (Rogers, Farinelli, Stranders, & Jennings, 2011; Zivan et al., 2017, 2020), evaluation in realistic applications (Ramchurn et al., 2010) and computational complexity (Macarthur et al., 2011; Kim & Lesser, 2013). Our work contributes to this ongoing effort by extending the applicability of Max-sum to teams of mobile sensing agents.

While Max-sum has been shown to efficiently solve DCOPs and has been used to coordinate sensors' movements, one of its major drawbacks is the run-time required for function-nodes to produce messages, which is exponential in the arity of the constraint that the function-node represents. Multiple attempts to overcome this drawback have been published in the last decade. Some of them, including the methods proposed by (Macarthur et al., 2011; Pujol-Gonzalez, Cerquides, Meseguer, Rodriguez-Aguilar, & Tambe, 2013), were implemented in the Max-sum version that was proposed for solving DCOP_MST in (Yedidsion, Zivan, & Farinelli, 2014). Others, which were proposed recently, make an immense reduction of the computational cost in such scenarios (Khan, Tran-Thanh, Ramchurn, & Jennings, 2018; Chen, Jiang, Deng, Chen, & He, 2019). In our work, we prove that such exponential computation is not required, and therefore these methods, which are most useful in standard scenarios, are less relevant.

Creating collision-free paths is the main objective of Multi-Agent Path Finding (MAPF) (Sharon, Stern, Felner, & Sturtevant, 2015; Atzmon, Stern, Felner, Wagner, Barták, & Zhou, 2020; Stern et al., 2019), which focuses on creating paths for $n$ agents on their way to their targets while avoiding spatial conflict between the agents, and optimizing some global criteria, usually minimizing the total travel distances or minimizing the makespan. Although MAPF algorithms and CAMS concentrate on collision avoidance, MAPF's difference from our problem is threefold: (1) The main objectives. While we care to optimize target coverage as a cooperative effort accounting for collision avoidance as an additional constraint, MAPF focuses on agents' paths to targets. (2) The basic model and means to solve the problem. CAMS solves DCOP_MST with the addition of collision avoidance constraints, an inherently-dynamic *distributed* constrained optimization problem requiring the agents to have no global knowledge of the world, and MAPF is a centralized problem, solved most commonly by centralized search-based methods. (3) DCOP_MST is designed to be relevant to highly dynamic scenarios and to scenarios in which agents have limited computation and communication capabilities. Thus, its design is myopic, and the agents' domains and constraints are updated following each move. MAPF on the other hand, centrally models the entire problem. Therefore solutions to MAPF problems cannot be applied in our setting. Note that distributed MAPF was recently mentioned as one of the open challenges in MAPF (Salzman & Stern, 2020). Although there have been attempts to provide solutions to this problem (e.g., (Pianpak, Son, Toups, & Yeoh, 2019)), those still remain irrelevant for solving DCOP_MSTs.

The last argument regarding the differences between DCOP_MST and MAPF is also relevant to MDP-based models (such as POMDP). These models are much more abstract and general and obviously can be used to represent any realistic problem. However, this generalization comes with a complexity cost that limits their usefulness to very small problems. Thus, studies that target specific domains have proposed useful models and algorithms that are designed for the specific domain. Some examples are MAPF, Proactive DCOP, and DCOP_MST (Sharon et al., 2015; Hoang, Fioretto, Hou, Yeoh, Yokoo, & Zivan, 2022; Zivan et al., 2015).

Finally, collision avoidance, being one of the fundamental requirements of a robotic system, is vastly explored in multi-robot systems (Hoy et al., 2015). The main focus in *decentralized* collision avoidance methods is on providing means for locally preventing collisions by considering the other robots as mobile obstacles, or suggesting local coordination schemes yielding collision-free paths (Serra-Gómez, Brito, Zhu, Chung, & Alonso-Mora, 2020; Tabasso, Cichella, Mehdi, Marinho, & Hovakimyan, 2021; DeCastro, Alonso-Mora, Raman, Rus, & Kress-Gazit, 2018). In our case collision avoidance is intertwined with the target coverage task, necessitating the creation of a method that considers both for yielding an optimal team behavior, which is the essence of CAMS.

## 3. Background

The DCOP model is commonly used for representing and solving coordination problems related to sensor networks (Farinelli et al., 2014; Nguyen et al., 2014) and mobile sensor networks (Stranders et al., 2009; Taylor et al., 2010; Zivan et al., 2015). To the best of our knowledge, none of these studies addressed the possibility of collisions between mobile sensors. A vast amount of research has been invested in recent years in modeling and solving multi-agent path-finding problems (MAPF) (Sharon et al., 2015; Atzmon et al., 2020). MAPF considers scenarios where a strong computing system (mostly centralized) can compute paths for all agents from their start position to their goal states while avoiding collisions. This is in contrast to scenarios on which we focus in this study,

where the team of sensors is composed of entities with low computing and sensing abilities, that only compute a small number of steps ahead. An attempt to use the DCOP model and algorithms for collision avoidance of ships was presented in (Hirayama et al., 2019), where the distributed stochastic search algorithm (DSSA) was used to prevent ships from selecting colliding routes. Distributed stochastic algorithms (DSA) are synchronous local search algorithms in which agents hold assignments and make greedy attempts to improve them, subject to a stochastic replacement decision. Several versions of DSA were found to be inferior to Max-sum_MST in (Yedidsion et al., 2014, 2018). Nevertheless, we compare the performance of our proposed algorithm with DSSA and a collision-avoiding version of DSA (CADSA) in our empirical study.

In this section, we provide the necessary background on DCOP, Max-sum, the DCOP_MST model and Max-sum_MST.

### 3.1 Distributed Constraint Optimization

A distributed constraint optimization problem (DCOP) (Stranders et al., 2009) 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. 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 function $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}_+ \cup \{0\}$ 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 control. Furthermore, agents have knowledge only of the constraints involving variables that they control. 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.

### 3.2 Standard Max-sum

Max-sum (Farinelli et al., 2008) operates on a *factor graph*, which is a bipartite graph including nodes that represent variables and constraints (Kschischang, Frey, & Loeliger, 2001). Each variable-node representing a DCOP variable is connected to all function-nodes that represent constraints, in which it is involved. 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: $Q^i_{X \rightarrow F} = \sum_{F' \in F_X, F' \neq F} R^{i-1}_{F' \rightarrow X} - \alpha$ where $Q^i_{X \rightarrow F}$ 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 $R^{i-1}_{F'\to X}$ is the message sent to variable-node $X$ by function-node $F'$ in iteration $i-1$. $\alpha$ 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. We will use the smallest utility in the vector as $\alpha$, thus, each message sent will include at least one belief equal to zero.

A message $R^i_{F\to X}$ sent from a function-node $F$ to a variable-node $X$ in iteration $i$, includes for each value $x \in D_X$:

$$max_{PA_{-X}} u(\langle X, x\rangle, PA_{-X})$$

where $PA_{-X}$ is a possible combination of value assignments to variables involved in $F$, not including $X$. The term $u(\langle X, x\rangle, PA_{-X})$ represents the utility 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 utility 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 utility 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: $\hat{x} = argmin_{x\in D_X} \sum_{F\in F_X} (R^k_{F\to X})_x$.

Assuming there are no tied beliefs, Max-sum converges in a linear number of iterations to the optimal solution when solving problems represented by a tree-structured factor graph (Pearl, 1988).[1] When it operates on a single cycle factor graph, it will either converge to the optimal solution or reach a state in which it repeatedly replaces assignments of the same (different) values (Forney, Kschischang, Marcus, & Tuncel, 2001).

### 3.3 The DCOP_MST Model

DCOP_MST includes agents $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ physically situated in the environment, which is modeled as a metric space. Each agent $A_i$ controls one variable, denoted by $cp_i$, that represents its current position. Time is discretized into an indeterminate series of time-steps, and the maximum distance $A_i$ can travel in a single time-step is defined by its *mobility range*, $mr_i$. Therefore, the domain of $cp_i$ contains all locations within $mr_i$ from it; consequently, once the agent moves, the content of its variable's domain changes. A change in the content of some variables' domains can induce a constraint change. The agents have limited heterogeneous sensing ranges, where $sr_i$ denotes the sensing range of agent $A_i$, and each agent can only provide information on targets within its sensing range. Moreover, agents may also differ in the quality of their sensing abilities, a property termed as their *credibility*. The credibility of agent $A_i$ is denoted by $cred_i \in R^+$, with higher values indicating better sensing abilities. 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 joint credibility required for that point to be adequately sensed. Thus, a target $T_j \in \mathcal{T}$ is a point $p$ with $ER(p) > 0$.

Agents whose current position is within the sensing range of target $T_j$ are said to *cover* it and the *remaining coverage requirement* $cr_j$, is $ER(T_j)$ diminished by the joint credibility of the agents currently covering the target, with a minimum value of zero. The coverage of sensors aiming to apply to the environmental requirement of a target is not accumulated. Thus, if a target requires the coverage of more than one sensor, they must simultaneously place themselves in the sensing

1. Ties can be avoided by adding for each variable-node a unary constraint with extremely small random utilities (Farinelli et al., 2008)

range from it. Denoting the set of agents within the sensing range of a point $p$ by $sr(p)$, this is formalized as $cr(p) = \max\{0, ER(p) \ominus F(sr(p))\}$, where $F$ is the joint credibility function that combines the credibility of neighboring agents and $\ominus : \mathcal{R} \times \mathcal{R} \rightarrow \mathcal{R}$ is a binary operator that decreases the environmental requirement by the joint credibility. For simplicity we will assume that $F(sr(p)) = \sum_{A_i \in sr_{(p)}} cred_i$ and that $\ominus$ is a standard subtraction (Yedidsion et al., 2018). The global goal of the agents is to position themselves so to minimize $\sum_{T_j \in \mathcal{T}} cr(T_j)$. Such a minimization problem is NP-hard (Wang, Cao, Berman, & Laporta, 2003).

### 3.4 Distributed Stochastic Algorithm (DSA)

The distributed stochastic algorithm (DSA) is a simple synchronous search algorithm for solving DCOPs (Zhang, Xing, Wang, & Wittenburg, 2005), which was applied to DCOP_MST, extended to avoid collisions, and which we use as a benchmark for comparison. In DSA, after an initial iteration in which agents select a value assignment for their variable (randomly according to (Zhang et al., 2005)), agents perform a sequence of iterations until some termination condition is met. In each iteration, an agent sends its value assignment to its neighbors and receives theirs. Versions of DSA differ in the stochastic method agents use to decide on whether to replace their value assignments. The algorithm uses a stochastic parameter $0 < p \leq 1$ in order to make this decision. If an agent in DSA cannot improve its current state (or keep the same cost, depending on the version used) by replacing its current value assignment, it does not replace it. Otherwise, it replaces its value assignment with probability $p$.

### 3.5 Convergence Properties

Belief propagation (in general and specifically Max-sum) converges in a linear number of iterations to an optimal solution when the problem's corresponding factor-graph is acyclic (i.e., have a tree-structured factor-graph) (Pearl, 1988). For a single-cycle factor-graph, we know that if belief propagation converges, then it is to an optimal solution (Forney et al., 2001; Weiss, 2000). Moreover, when the algorithm does not converge, it periodically changes its set of assignments. To explain this behavior, Forney et al. (2001) show the similarity of the performance of the algorithm on a cycle to its performance on a chain, whose nodes are similar to the nodes in the cycle, but whose length is equal to the number of iterations performed by the algorithm. One can consider a sequence of messages starting at the first node of the chain and heading towards its other end. Each message carries beliefs accumulated from utilities added by function-nodes. Each function-node adds a utility to each belief, which is the constraint value of a pair of value assignments to its neighboring variable-nodes. Each such sequence of utility accumulation (route) must at some point become periodic, and the maximal belief would be generated by the maximal periodic route. If this periodic route is consistent (i.e., the set of assignments implied by the utilities contain a single value assignment for each variable), then the algorithm converges. Otherwise, it does not.

We denote by a *path* the sequence of entries in the utility tables of the function-nodes along the route, which are accumulated in order to generate a belief.

### 3.6 Run-Time Complexity Analysis

The overhead in run-time complexity of CAMS (in comparison 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 (therefore there was no need to use recently published methods for reducing the computation of function-nodes in Max-sum (Khan et al., 2018; Chen et al., 2019)). On the other hand, while target function-nodes may have more than two neighbors, we will prove in Theorem 1 that the calculation is redundant, and thus, there is no need for these function-nodes to perform exponential computation. The computation performed by a variable-node in each iteration of the algorithm is the addition of the vectors received from its neighbors, for each message it sends. Thus, for an agent performing the computation of a single variable-node with $k$ neighbors, the run-time complexity in each step of the algorithm is $k2(k-1) = O(k^2)$.

### 3.7 Applying Max-sum to DCOP_MST

Max-sum_MST, the Max-sum version applied to DCOP_MST, implements an iterative process in which in each step the agents construct a factor-graph based on their current locations, run the Max-sum algorithm for a number of iterations, [2] and move according to the solution provided by the algorithm. In the next step, a new factor-graph is generated considering the new locations of the agents (Yedidsion et al., 2018).

A message sent from function-node $F$ representing target $T \in \mathcal{T}$ to a variable-node $X$, $R_{F \to X}$, includes two utilities, one for locations from which the sensor covers the target, and one for locations from which it does not (as in fast Max-sum (Macarthur et al., 2011)). A factor-graph in Max-sum_MST is generated using the function meta reasoning (FMR) method (Yedidsion et al., 2018). It is used when there are more neighbors than required for covering a target in order to avoid symmetry (i.e., prevent a situation in which all neighboring agents decide to cover some target, or alternatively, all decide not to cover it). Consider a step $i$ in which the factor-graph $FG^i$ was generated based on the locations of sensors selected in step $i-1$. Denote by $n(T)^i$ the set of neighboring sensors of the target $T \in \mathcal{T}$ in $FG^i$. Denote by $r(T)^i$ a subset of $n(T)^i$ and let $cred_{r(T)^i} = \sum_{A_j \in r(T)^i} cred_i$. The function-node $F$ representing target $T$ selects the minimal subset $r(T)^i$ for which $ER(T) \leq cred_{r(T)^i}$, and removes the edges connecting it in the factor-graph to the sensors in $n(T)^i \setminus r(T)^i$. We will denote this set (the set of neighbors of $T$ resulting from the FMR procedure in step $i$) by $N(T)^i$.

FMR is a first step to avoid symmetry, but it is not enough: If the accumulated credibility of target $T$'s neighbors $N(T)^i$ is larger than its environmental requirements $ER(T)$, there is still a need to break symmetry in order to avoid sending distorted requirements to its neighbors (either high for all neighbors or low for all of them). Yedidsion et al. (2018) suggested an ordered value propagation (OVP) approach, in which the neighbors are ordered and the utility for the last neighbor in the order is reduced. We propose a more balanced approach in the following section.

## 4. Problem Formalization

The problem that we address in this paper is DCOP_MST, as described in Section 3.3 and in (Zivan et al., 2015; Yedidsion et al., 2018), with the addition of the consequences of collisions. Thus, we formalize CA-DCOP_MST, which includes all elements of DCOP_MST with the addition of collision constraints.

One element that is implicitly defined in DCOP_MST is the time intervals. In DCOP_MST the domains and constraints are determined according to the location of the agents at the current

2. In order to avoid confusion we use the term step for the process of selecting a physical by the robots and the term iteration for the calculation-steps that are performed by the agents as part of the algorithm within a step.

time. Thus, after the following movement, in a future time, the location of the agents changes and with it the domains for their variables and their constraints. We add to the CA-DCOP_MST model these time intervals (or steps) explicitly. The time intervals are ordered in a discrete finite sequence $\{t_0, t_1, t_2, ..., t_f\}$, such that in each such time $t_k$, each agent $A_i$ is located at $cp_i^{t_k}$.

Following the vast amount of existing work on multi-agent path finding (MAPF), we add to DCOP_MST a set of hard constraints that prevent two agents from taking the same position at the same state.

Formally, CA-DCOP_MST includes a set of binary collision constraints $CC$, where $CC_{ij}$ is the hard constraint between agents $A_i$ and $A_j$. $CC_{ij}$ prevents agent $A_i$ and agent $A_j$ from selecting the same position in the same time interval, i.e. $cp_i^{t_k} \neq cp_j^{t_k}$ and from switching adjacent positions, i.e., if agents $A_i$ and $A_j$ are located at $cp_i^{t_k}$ and $cp_j^{t_k}$ at time $t_k$, then it is not possible that $cp_i^{t_{k+1}} = cp_j^{t_k}$ and $cp_j^{t_{k+1}} = cp_i^{t_k}$ in iteration $t_{k+1}$.

In DCOP_MST, two agents are considered neighbors if after moving as close as possible towards each other in a single iteration, their sensing ranges overlap (Zivan et al., 2015), i.e., an agent $A_i$ is the neighbor of agent $A_j$ at iteration $t_k$ if and only if the distance between $cp_i^{t_k}$ and $cp_j^{t_k}$ is smaller than $mr_i + sr_i + mr_j + sr_j$. A $CC$ constraint at iteration $t_k$ exists between two agents if and only if they can move to the same location in iteration $t_{k+1}$ (notice that, since an agent can stay in its location, this definition covers both types of $CC$ constraints). Thus, the addition of $CC$ constraints does not change the agents' neighbor sets.

A solution to CA-DCOP_MST is an assignment (position) for each of the agents' variables $cp$, such that none of the $CC$ constraints are violated.

It is important to mention that, in contrast to MAPF, DCOP_MST (hence, also CA-DCOP_MST) is myopic, thus, $CC$ constraints exist only between agents that can select the same position in the next step. Like in the case of the target constraints and the content of the domains, following each movement of the agents, the $CC$ set is updated.

## 5. Collision Avoiding Max-sum (CAMS)

Collisions among mobile sensors may result in damaging the sensors, execution delay, or even the inability to perform the coverage task. Thus, we propose Collision Avoiding Max-sum (CAMS) for solving CA-DCOP_MST, which 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 step of Max-sum_MST (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.

In more detail, in a factor-graph generated in CAMS there are three types of function-nodes:
1) $FT_j$, a function-node representing a target $T_j$. The neighbors of the function are selected using FMR. However, instead of performing ordered value propagation, the targets adjust the utilities sent to their neighboring agents in a more balanced manner. Let $u_{ij}^{cov}$ denote the utility sent to neighbor $A_i$ by target $FT_j$ for covering it. The value of $u_{ij}^{cov}$ is defined as follows:

$$u_{ij}^{cov} = \begin{cases} ER_{FT_j}, & \text{if } ER_{FT_j} < cred_i \\ cred_i - \max\{0, \frac{\sum_{i' \in N(FT_j)} cred_{i'} - ER_{FT_j}}{|N(FT_j)|}\}, & otherwise \end{cases}$$

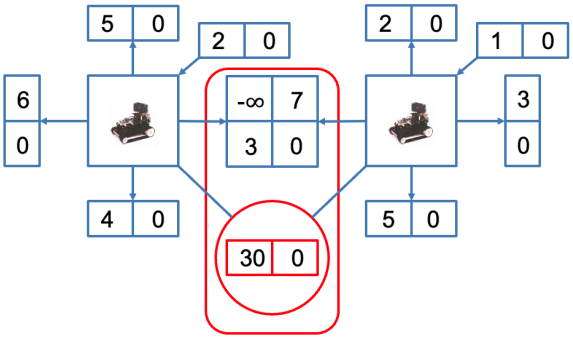


Figure 1: A factor-graph generated in CAMS, including two neighboring agents. Each can move in four directions or stay. Among the four options, three are of type $FL_i$ and one (the one between the agents) is of type $FL_{ij}$. The red circle represents a target function-node ($FT_i$), and it can be covered only from the location to which both agents can move.

We refer to this calculation of the neighbors' coverage utilities as *balanced utility adaptation* (BUA). The motivation that triggered the replacement of OVP by BUA is that a reduction of all the excess coverage from one single sensor may result in a low probability that this sensor will cover the target. BUA on the other hand generates a minor reduction for each of the covering sensors and thus, the outcome is more stable, as we demonstrate in our experimental study (see Section 7).
2)$FL_{(i)}$, a function-node representing a location to which only one agent can move. The corresponding constraint is unary. A random positive utility is selected for the option that $A_i$ selects this location (selected from the same range as the random utilities selected for the binary constraints described next) and zero for not selecting this location.
3)$FL_{(i,e)}$, a function-node representing a location $l$ to which mobile sensors $A_i$ and $A_e$ can move in this iteration. The utility for both mobile sensors for not selecting location $l$ is zero, for both 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 $ER_{FT_j}$[3]. In the case where one of the mobile sensors is located in $l$ (without loss of generality, assume this is $A_i$), then the option that $A_e$ moves to $l$ and $A_i$ moves to the current location of $A_e$ is also excluded by utility -∞, and thus, edge constraints for avoiding collisions are enforced. We emphasize that although there may exist scenarios in which more than two mobile sensors can move to the same location, $FL$ is defined as a binary constraint, and thus, if there are $k > 2$ mobile sensors that can select the same location, there will be an $FL$ for each pair of these $k$ mobile sensors.

Figure 1 presents an example of a factor-graph generated in some step of CAMS. It includes two mobile sensors, each with four possible locations to move to, and the option to stay in their current location. All function-nodes representing locations to which only one mobile sensor can move are of the second type. While the domain of each mobile sensor's current position variable 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

3. Random numbers are selected to avoid ties between desired options, as in (Farinelli et al., 2008)

| | $Comb_1$ | $Comb_2$ | ... | $Comb_{2^n}$ |
|---|---|---|---|---|
| $no\_cover$ | $u_1$ | $u_2$ | ... | $u_{2^n}$ |
| $cover$ | $u_1 + u^{cov}_{A,T}$ | $u_2 + u^{cov}_{A,T}$ | ... | $u_{2^n} + u^{cov}_{A,T}$ |

Table 1: Function-node message generation.

third type. It includes four options, one for both mobile sensors not selecting this location (zero utility), one for both selecting this location (minus infinity), and two with positive utilities for the cases which only one mobile sensor 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.

## 6. Properties of CAMS

In this section, we discuss properties of CAMS that affect its convergence, collision-free guarantees, and its run-time analysis.

### 6.1 Messages sent by Target Representing Function Nodes

To identify the properties of the factor-graphs on which CAMS is guaranteed to converge to the optimal (collision-free) solution and to analyze its run-time properties, we aim to prove that function-nodes in Max-sum_MST (when implementing FMR, and BUA or OVP) do not change the content of the messages they send throughout the algorithm run. This property of Max-sum_MST has not been reported, nor proven, in previous studies.

We use in our proof a table (as depicted in Table 1), which represents the calculations performed by a target representing function-node $FT$, when generating a message to be sent to mobile sensor $A$. This table includes *two* rows, one for positions from which $A$ covers the target and one for positions from which it does not. The columns represent combinations of position selections of all neighboring mobile sensors that can either select a position from which they cover the target or a position from which they do not. Thus, if there are $n$ such neighbors (not including $A$), the number of columns will be $2^n$. Each entry in the table includes the sum of the utility derived from the coverage, which the mobile sensors that selected a covering position in this column provided, and the relevant beliefs included in the messages received from the target's neighbors in the previous iteration.

**Theorem 1** *In every step of CAMS and of Max-sum_MST, each target representing function-node will send to its neighbors in every iteration of the Max-sum algorithm, a message including zero utility for not covering this target and $u^{cov}_{A,T}$ for covering it (here, $T$ is the target and $A$ is the neighbor to which the message is sent).*

**Proof:** Table 1 demonstrates the calculation performed by function-node $F_T$, representing target $T$, when generating a message to be sent to its neighboring mobile sensor $A$. The message will include two utilities (beliefs), one for covering the target and one for not. As stated above, each of the columns represents a combination of positions selected by the target's other neighbors. The

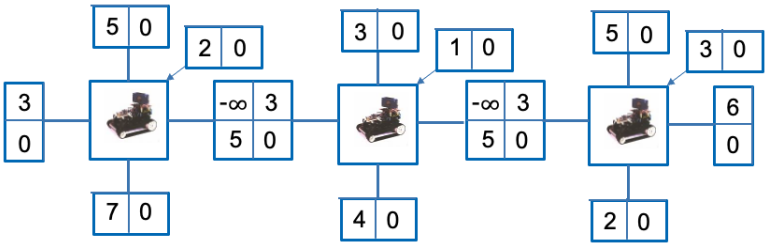


Figure 2: A tree-structured factor-graph including only location representing function-nodes.

target selects the largest utility in each row to be included in the message it sends to $A_i$, one for not covering the target and the other for covering it. For each column $j$, the utilities are $u_j$ for the first (non-covering) row, and $u_j + u^{cov}_{A,T}$ for the row representing a covering position of $A$. Thus, if for some column $j'$ and for all other columns $j \neq j'$, $u_{j'} > u_j$, then the message will include $u_{j'}$ for not covering the target and $u_{j'} + u^{cov}_{A,T}$ for covering it. Thus, after the reduction of $\alpha = u_{j'}$ the message sent will include $\langle 0, u^{cov}_{A,T} \rangle$. □

**The significance of Theorem 1** is that it explains previously published evidence regarding the instant convergence of Max-sum_MST (e.g., (Yedidsion et al., 2018)) in contrast to Max-sum's behavior on other benchmarks (Chen et al., 2018; Cohen et al., 2020).

### 6.2 Convergence Properties

We start the discussion by identifying factor-graph structures on which CAMS is guaranteed to converge to the optimal solution, and then we establish a more general property. Like in the case of standard Max-sum, these graph structures are interesting, not because they are common in realistic scenarios, but rather because we can establish theoretical properties for them (Forney et al., 2001; Weiss, 2000; Cohen, Lev, & Zivan, 2023). For realistic graph structures on which we do not establish theoretical results, we present empirical results in Section 7.

**Lemma 2** *In any step of CAMS, if the factor-graph includes no cycles, and a collision-free solution exists, the algorithm will converge to the optimal collision-free solution in a linear number of iterations.*

**Proof:** The factor-graph representation of this scenario has a tree structure and thus, Max-sum will converge in a linear number of iterations to an optimal solution (Pearl, 1988). This optimal solution[4] cannot include the selection of the same location by two or more agents, since the utility of such a mutual selection is $-\infty$. □

This proof is immediate, given the properties of belief propagation as established in (Pearl, 1988). We mention it just to indicate that the hard constraint function-nodes do not interfere with this property, as long as a solution that does not violate hard constraints exists. Note that if the current position of the agents is collision-free, then indeed such a solution exists, since the agents can choose to stay in their current locations.

4. Here by optimal solution we mean the collision-free solution that provides the smallest remaining coverage in this step.

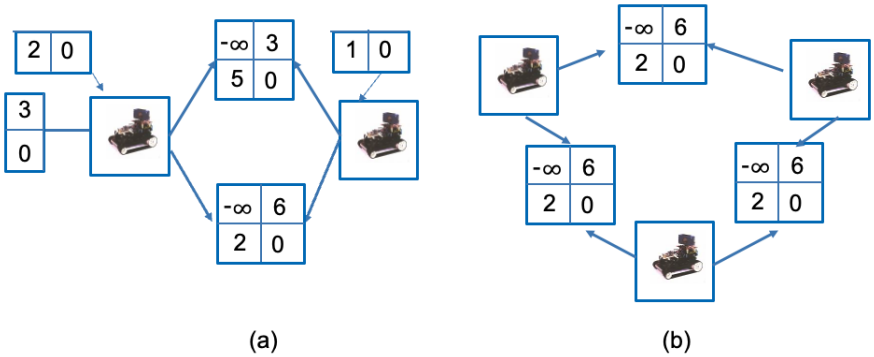


Figure 3: Single cycle factor-graphs including (a) two mobile sensors and (b) three mobile sensors.

Figure 2 presents an example of a tree-structured factor-graph including only location representing function-nodes. Note that additional function-nodes of the second type do not change the tree structure of the graph. In the optimal solution, the left mobile sensor moves down, the middle one moves to the left, and the one on the right moves to the right. The (optimal) utility derived from this solution is 18.

Tree-structured graphs include all scenarios in which adjacent mobile sensors have only one location that they both can move to. However, when there is more than one such location, or when more than two mobile sensors can move to the same location, the factor-graph includes a cycle (See examples for two such scenarios in Figure 3).

**Lemma 3** *In any step of CAMS, if the factor-graph includes a single cycle with two or more location representing function-nodes of the third type $FL_{(i,j)}$, and a collision-free solution exists, the algorithm converges to a collision-free optimal solution in a pseudo linear number of iterations.*

**Proof:** The factor-graph representation of such scenarios includes a single cycle with at least two $FL_{(i,j)}$ function-nodes. This type of function-node has four entries in the utility table, one of them includes minus infinity utility. According to (Forney et al., 2001), when belief propagation is applied to a single cycle graph, it converges to the optimal solution if and only if the optimal (maximal in our case) repeated path is consistent. The only way to generate a maximal inconsistent path in such cycles including two or more function-nodes with four entry utility tables is when the maximal path visits alternately entries of opposing directed diagonals in the utility tables (see proof in the following section). Since in all utility tables, the entry representing the movement of both mobile sensors to the represented location is equal to minus infinity, it is impossible to generate a maximal path including opposing directed diagonals. Thus, the maximal path must be consistent. The number of iterations depends 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. □

**Lemma 4** *In any step of CAMS, if the Max-sum algorithm has converged, two sensors will not move to the same location.*

**Proof:** For two sensors to move to the same location, the nodes representing them $X_i$ and $X_j$ in the factor-graph must have a mutual location function-node $F_{ij}$ neighbor. Assume without loss of generality that $F_{ij}$ includes the costs:

- $-\infty$ for both agents selecting it as their location.
- $x$ for agent $X_i$ selecting this location while $X_j$ does not.
- $y$ (without loss of generality we assume that $x > y$) for agent $X_j$ selecting this location while $X_i$ does not.
- zero if they both do not select this location.

These selections are demonstrated in Figure 4.

Further assume that the messages sent from $X_i$ and $X_j$ to $F_{ij}$ included: $\langle c_{i_a}, c_{i_b} \rangle$ and $\langle c_{j_a}, c_{j_b} \rangle$, respectively with the first cost in each pair referring to selecting the location represented by $F_{ij}$ and the second for not selecting it.

We note that:

- $\langle c_{i_a}, c_{i_b} \rangle$ is the sum of messages that $X_i$ receives from all its other neighbors (not including $F_{ij}$) after convergence.
- $\langle c_{j_a}, c_{j_b} \rangle$ is the sum of messages that $X_j$ receives from all its other neighbors after convergence.
- and we denote by $\langle f_{i_a}, f_{i_b} \rangle$ and $\langle f_{j_a}, f_{j_b} \rangle$ the costs that $F_{ij}$ sends to $X_i$ and $X_j$ respectively.

Thus, $X_i$ will choose the location represented by $F_{ij}$ if and only if $c_{i_a} + f_{i_a} > c_{i_b} + f_{i_b}$.
Similarly, $X_j$ selects to move to the location represented by $F_{ij}$ if and only if $c_{j_a} + f_{j_a} > c_{j_b} + f_{j_b}$.
The costs $f_{i_a}$ and $f_{i_b}$ are calculated as follows:

$f_{i_a} = c_{j_b} + x$
$f_{i_b} = \max\{c_{j_a} + y, c_{j_b}\}$
$f_{j_a} = c_{i_b} + y$
$f_{j_b} = \max\{c_{i_a} + x, c_{i_b}\}$
Thus, a collision occurs when:
$c_{i_a} + c_{j_b} + x > c_{i_b} + \max\{c_{j_a} + y, c_{j_b}\}$ and $c_{j_a} + c_{i_b} + y > c_{j_b} + \max\{c_{i_a} + x, c_{i_b}\}$

Assume that: $c_{j_b} > c_{j_a} + y$ , then $c_{i_a} + c_{j_b} + x > c_{i_b} + c_{j_b}$ and therefore $c_{i_a} + x > c_{i_b}$
Hence in the second inequality we have $c_{j_a} + c_{i_b} + y > c_{j_b} + c_{i_a} + x$
and therefore we can replace on the right side: $c_{i_a} + x$ *by* $c_{i_b}$ and get that
$c_{j_a} + c_{i_b} + y > c_{j_b} + c_{i_b} \Rightarrow c_{j_a} + y > c_{j_b}$, which contradicts our assumption.

Similarly, when $c_{i_b} > c_{i_a} + x$ then $c_{j_a} + c_{i_b} + y > c_{j_b} + c_{i_b} \Rightarrow c_{j_a} + y > c_{j_b}$
Thus, in the first inequality we have $c_{i_a} + c_{j_b} + x > c_{i_b} + c_{j_a} + y$
We can replace: $c_{j_a} + y$ with $c_{j_b}$ and get: $c_{i_a} + c_{j_b} + x > c_{i_b} + c_{j_b} \Rightarrow c_{i_a} + x > c_{i_b}$,
which again contradicts our assumption.

We are left with one case in which:
$c_{i_a} + c_{j_b} + x > c_{i_b} + c_{j_a} + y$ and $c_{j_a} + c_{i_b} + y > c_{j_b} + c_{i_a} + x$

It is easy to see that these two contradict each other, concluding the proof □

To demonstrate more intuitively the property proved in Lemma 4, Figure 4 illustrates two agents, $X_i$ and $X_j$, a location representing function node $F_{ij}$ to which both agents can move, and the agents' messages.

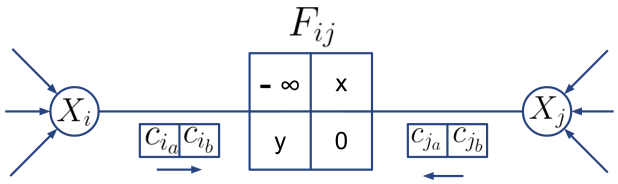


Figure 4: One $F_{ij}$ (function-node for location) and two agents

The messages sent by $F_{ij}$ are presented in Figure 5. There are two options for each message, depending on the largest candidate for not choosing the location represented by $F_{ij}$. For example, if $c_{i_a} + x > c_{i_b}$ then the message from $F_{ij}$ will be $\langle c_{i_b} + y, c_{i_a} + x \rangle$, otherwise it will be $\langle c_{i_b} + y, c_{i_b} \rangle$. The location selected by each agent is based on the sum of all the messages that it received in the last

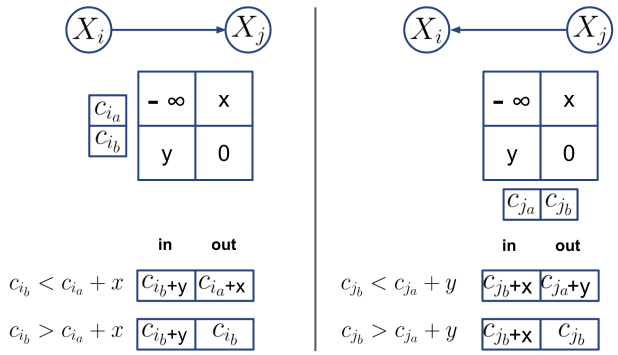


Figure 5: Messages sent by $F_{ij}$

iteration. Those sums are presented in Figure 6. In Figure 7 four possible variants of calculations are performed by the agents when selecting their location, depending on the content of the messages received in the last iteration. In variants II, III, and IV the agents would select the options marked in red. In option I there are two possible outcomes (marked with purple and orange). None of the combinations results in agents taking the location represented by $F_{ij}$ simultaneously.

**Theorem 5** *In every step of CAMS, if the Max-sum algorithm converges, the agents do not collide.*

Immediate following Lemma 4. It is important to mention that, while we do not prove that Max-sum always converges in CAMS, our empirical results indicate that this is indeed the case.

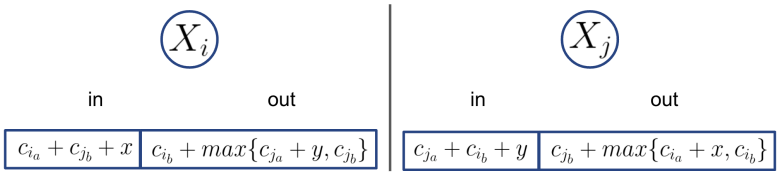


Figure 6: Sum of messages by each agent

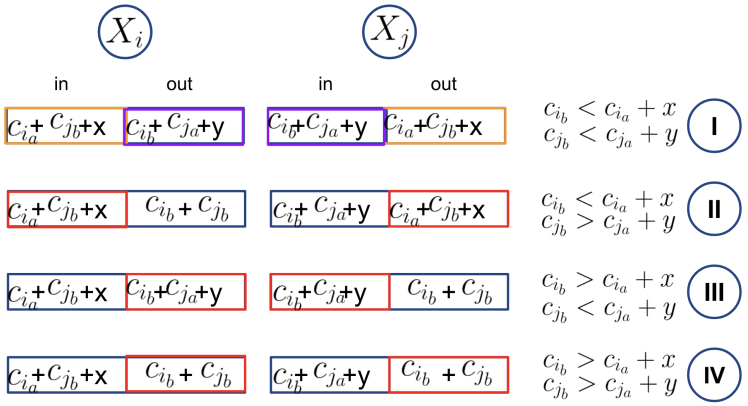


Figure 7: Location Selection Calculations

### 6.3 Consistent and Inconsistent Paths

In the proof of Lemma 3 we stated that in a single cycle graph including two or more function-nodes representing binary constraints with four entries in their utility table, the maximal path can be inconsistent if and only if this path visits entries that are part of opposite direction diagonals. Consider such a cycle (as depicted in Figure 8 (a)). Assume each agent can select value assignments $v_1$ or $v_2$. Thus, the four entries represent the pairs of assignments $\langle v_1, v_1 \rangle$ on the top left, $\langle v_2, v_1 \rangle$ on the bottom left, $\langle v_1, v_2 \rangle$ on the top right and $\langle v_2, v_2 \rangle$ on the bottom right. Without loss of generality, we will follow a clockwise path that starts with the top left entry in the bottom utility table in the example depicted in Fig 8 (a). A consistent path will have both agents take $v_1$, i.e., the assignment will include $\langle v_1, v_1 \rangle$ and the path will visit the left top entries in both utility tables until the algorithm terminates resulting in the path $15 \rightarrow 1 \rightarrow 15 \rightarrow 1$.... On the other hand, an inconsistent path includes shifts of assignments for each variable. However, since it is a path of a route, this shift is done alternately by the agents. Thus, after visiting $\langle v_1, v_1 \rangle$ in the bottom utility table, the path shifts to $\langle v_1, v_2 \rangle$ in the top utility table, then it shifts to $\langle v_2, v_2 \rangle$ in the bottom utility table, to $\langle v_2, v_1 \rangle$ in the top utility table and back to $\langle v_1, v_1 \rangle$ in the bottom. The path visited is $15 \rightarrow 2 \rightarrow 10 \rightarrow 6 \rightarrow 15$.... One of these two passes is maximal (since the accumulated utility of all other consistent and inconsistent paths is much lower). To compare we need to normalize by length. The inconsistent path visits four entries their sum is 33. The consistent path visits two entries, and their accumulated utility is 16. To normalize we will visit this path twice and get a utility of 32. Thus, the maximal path here is the inconsistent path. However, in the example depicted in Figure 8 (b) we increased the

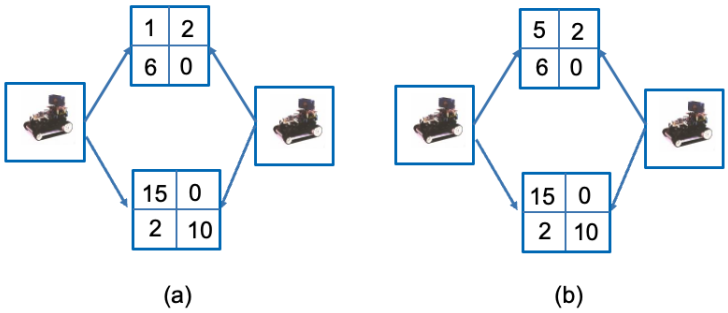


Figure 8: A single cycle factor-graph with two four-entry function nodes.

utility in the entry for $\langle v_1, v_1 \rangle$ (top left) in the top utility table from 1 to 5. Now, the accumulated normalized utility of the maximal consistent path is 40, while the maximal accumulated utility of an inconsistent path remains 33. Thus, when solving the factor graph depicted in Figure 8 (b), Max-sum would converge to the solution $\langle v_1, v_1 \rangle$, while when solving the factor graph depicted in Figure 8 (a) it would not converge.

A similar observation to the one we stated following Lemma 2 can be noted here as well. Additional unary constraints (function-nodes of the second type) would not affect the correctness of Lemma 3, since regardless of their content, they cannot overcome the minus infinity utilities that prevent an inconsistent maximal path.

Note that factor-graphs similar to the ones depicted in Figure 3(a) are generated when two mobile sensors can both move to two different locations and that factor-graphs similar to the one depicted in Figure 3(b) are generated when three mobile sensors can move to a single location. When more than three mobile sensors can move to a single location, the representing factor-graph will include more than one cycle.

In order to state the final Theorem in this section, we introduce the following definition: The *Underlying location factor-graph* (ULFG) is the factor-graph that is generated in a step of CAMS, after removing all target representing function-nodes and all the edges connecting them to the mobile sensor variable nodes. Thus, all the function-nodes in a ULFG represent locations.

**Theorem 6** *In any step of CAMS in which a factor-graph G is being solved by Max-sum, if the ULFG of G is tree-structured or includes a single cycle, then Max-sum is guaranteed to converge on G to the optimal solution (collision-free, if such a solution exists) in a pseudo linear number of iterations.*

**Proof:** According to Theorem 1, the target representing function-nodes in $G$ constantly send the same messages. Thus, they act as if they are unary constraints (function-nodes with a single variable-node neighbor) and are not affected by the messages sent to them. According to (Forney et al., 2001; Zivan et al., 2020), Max-sum will converge if and only if the maximal path is consistent. As stated in the observations following Lemmas 2 and 3, unary constraints cannot change the fact that the maximal path in these graphs cannot be inconsistent. □

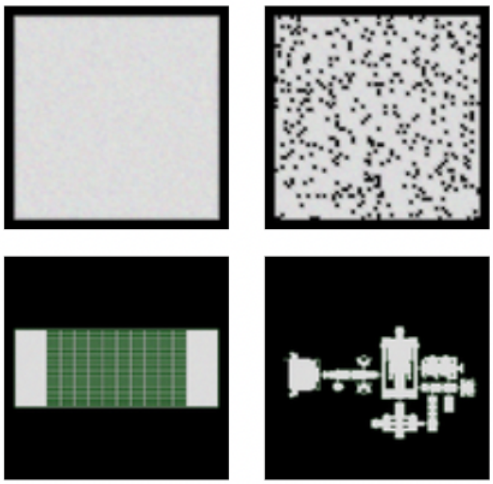

Figure 9: Left to right, top to bottom: empty, random, warehouse, and game environments.

## 7. Experimental Evaluation

To evaluate the performance of CAMS, we designed two simulators, a software simulation and a hardware simulator including robots. The first set of experiments was performed in software simulation, implemented in Python[5], allowing for rigorous evaluation of CAMS compared to existing algorithms. We used a MacBook Air with an Apple M1 chip and 8GB of RAM. In the second set of experiments, we have fully implemented CAMS on a physical multi-robot system that included 3 Hamster robots (Cogniteam, 2020). We report here a subset of the results (similar trends were observed when other parameters were used as well, and the results were not included to avoid redundancy).

### 7.1 Software Simulation

We used in our experiments a classical Grid-based MAPF benchmark (Stern et al., 2019; Sharon et al., 2015). This publicly available[6] benchmark consists of 24 different grids that were based on maps of real cities, video games, open areas with and without random obstacles, maze-like environments, and room-like spaces. Those grids are based on the MovingAI pathfinding repository[7] (Sturtevant, 2012).

Our software simulation included four environments from the aforementioned benchmark: empty ($48 \times 48$), random ($32 \times 32$), warehouse ($63 \times 161$), and game ($180 \times 251$). A demonstration of the environments is presented in Figure 9.

The number of agents included in each of the scenarios was 20 and the number of targets was 10. For each environment, we tested the algorithms in both static and dynamic settings. In the static settings, the number of targets and their locations were constant, while in the dynamic setting targets randomly switched to new positions every $t$ steps (we used $t = 40$). The ER values of all targets were

5. https://github.com/Arseni1919/dcop_simulator_5
6. https://movingai.com/benchmarks/mapf.html
7. https://movingai.com/benchmarks/grids.html

100, the sensing range of all sensors, $sr_i$, was set to 5, and the credibility of each sensor was 22. The positive utilities of location function-nodes were selected randomly from the range $[10^{-10}, 10^{-5}]$. In each step of the algorithm, each mobile sensor could either move to one of the adjacent locations or stay in its current location. For each environment, the experiments included 20 scenarios, and we report the average remaining coverage requirement and the accumulated number of collisions obtained by each algorithm solving these scenarios. Each algorithm performed 200 steps, where in each step the mobile sensors selected locations.

We compared CAMS with six other algorithms: **(1)** Max-sum_MST (including FMR and OVP), in which the mobile sensor movements were not affected by collisions; **(2)** Max-sum_MST with breakdowns, in which colliding agents exhibited a breakdown and stopped moving, but kept on sensing and communicating with other sensors; **(3)** DSA_MST, the standard DSA version described in (Yedidsion et al., 2018); **(4)** DSA_MST with breakdowns. Like in the Max-sum_MST version, the results of a collision in this version was that the agents stopped moving but kept on sensing; **(5)** CADSA, a collision avoiding version of DSA_MST. We ranked the mobile sensors according to their indexes. Each mobile sensor updated its neighbors before moving to a new location. A mobile sensor did not move to a new location if a different mobile sensor with a higher rank reported that it plans to move to the same location. **(6)** DSSA, the distributed stochastic search algorithm, designed to avoid collisions between ships (Hirayama et al., 2019). DSSA allows mobile sensors to keep suggesting locations they intend to move to while checking for collisions until they converge to a collision-avoiding decision in each step. **(7)** Random walk, serves as a baseline for the number of expected collisions.

CAMS and Max-sum_MST performed 10 iterations of Max-sum in each step before the mobile sensors selected their locations. The remaining coverage requirement in each step was calculated as $\sum_{T_j \in T} cr(T_j)$. We performed t-tests with $p < 0.01$ between the remaining coverage results at the 200 steps for each algorithm solving each scenario, to evaluate statistical significance when comparing the average results produced by the different algorithms and report standard deviations as well (presented in the Tables 2 and 3 in the appendix).

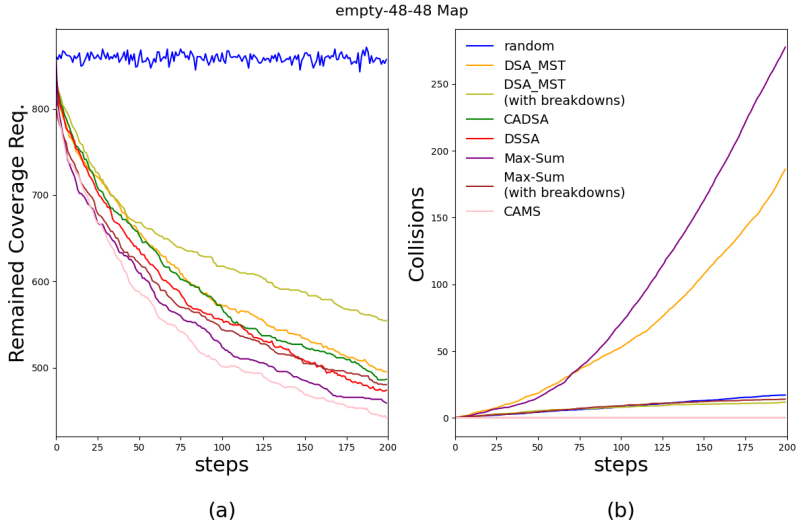


Figure 10: *empty* environment with *static* targets

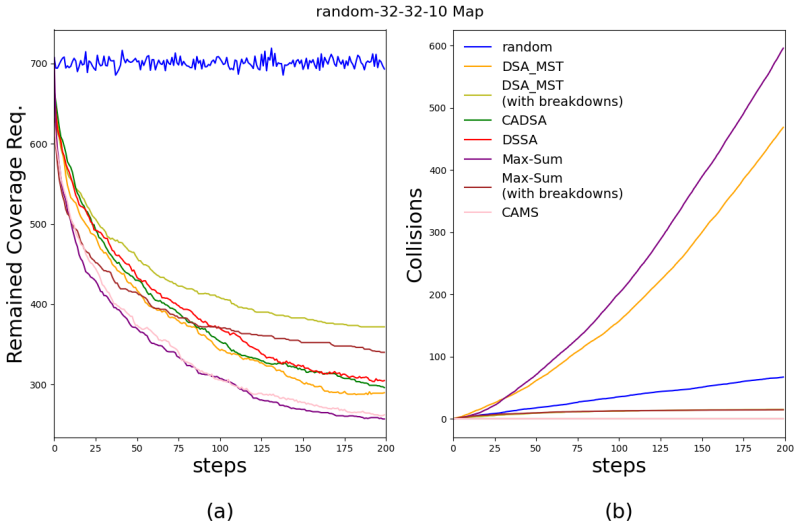


Figure 11: *random* environment with *static* targets

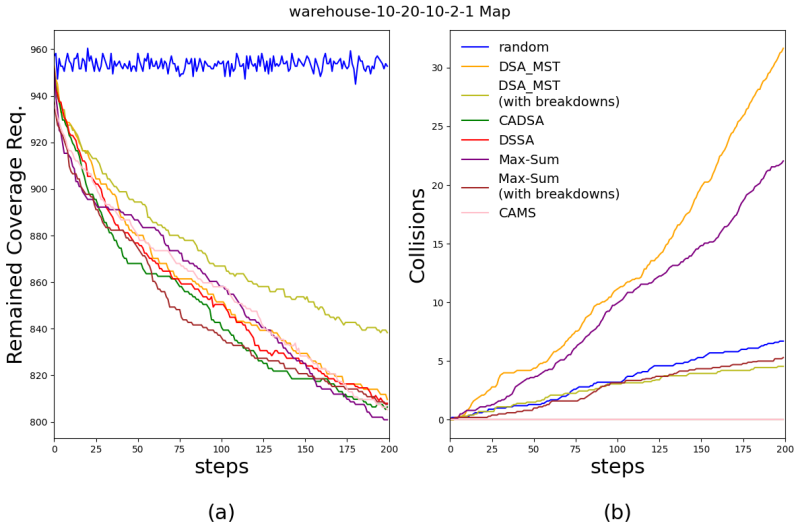


Figure 12: *warehouse* environment with *static* targets

Figures 10(a), 11(a), 12(a) and 13(a) present the remaining coverage requirement of the sensors performing the different algorithms as a function of the number of steps in the *static* settings. Each graph presents the results of the algorithm, performed in a different environment. Nevertheless, the trends are similar in all environments. It is clear that both CAMS and Max-sum_MST had a significant advantage over random walk and all the DSA versions. Since the mobile sensors in Max-sum_MST do not avoid collisions, they are less restricted and therefore the resulting coverage is significantly better than the results of this algorithm in the experiments that included breakdowns. Nevertheless, CAMS performed competitively in comparison to the Max-sum_MST version that did not exhibit breakdowns, that is, CAMS was able to find collision-free solutions that their coverage results are similar (in the *empty* and *random* environments) or significantly better (in the *warehouse* and *game* environments) than the solutions produced by the best algorithm that ignored collisions.

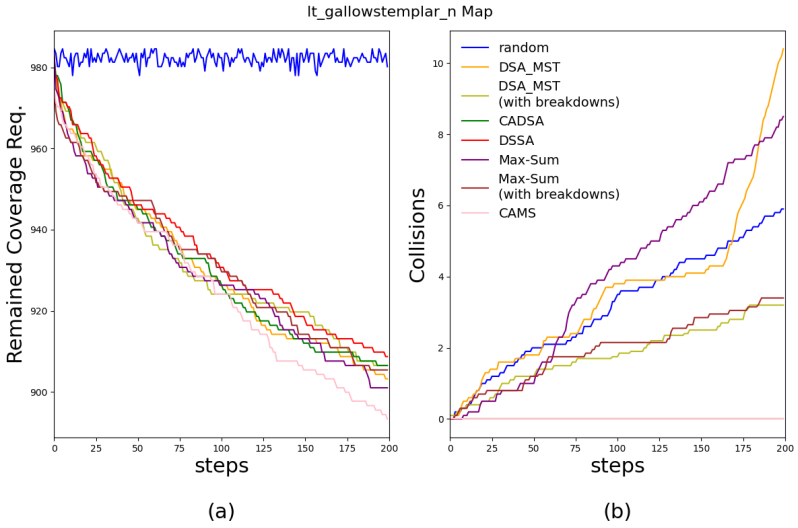


Figure 13: *game* environment with *static* targets

The different versions of DSA produced solutions with similar quality, and the performance of all of them was inferior to the Max-sum versions. When considering the Max-sum_MST version and the DSA version that included breakdowns, it is interesting to note that: (1) both were inferior, compared to the versions of those algorithms that did not exhibit breakdowns and (2) they demonstrate a similar advantage of Max-sum over DSA as demonstrated in the experiments with the other versions of those algorithms. This advantage is more apparent in the empty, random, and warehouse scenarios. The game environment was also unique in that DSA versions performed similarly to Max-sum_MST versions. However, CAMS significantly outperformed them.

Figures 10(b), 11(b), 12(b) and 13(b) present the number of accumulated collisions for each algorithm in these experiments, as a function of the number of steps. Clearly, the algorithms that do not avoid collisions exhibit more collisions than random walk. This can be explained by the attraction of mobile sensors to locations from which targets can be covered. The more open-spaced environments, e.g., the empty environment, result in more collisions. We assume that this is a result of the environmental obstacles, e.g., a corridor in a warehouse, that prevent agents that are located on different sides of the obstacles from colliding. CAMS, as well as CADSA and DSSA, do not exhibit any collisions. The versions of Max-sum_MST and DSA_MST which include breakdowns, exhibit fewer collisions than the versions that do not. This is because each sensor can experience only a single collision in these versions, while in the no-breakdown versions, they can exhibit many.

Figures 14, 15, 16 and 17 present the remaining coverage (a) and the number of accumulated collisions (b) of the different algorithms in the *dynamic* setting. In *empty* and *random* environments, CAMS significantly outperforms all other algorithms in terms of remaining coverage except for Max-sum_MST which ignores collisions. In addition, the gap between algorithms grows with the progression of steps, while CAMS continues to successfully adapt itself to periodic changes and maintains a consistent advantage over the other algorithms. In more complex environments such as *warehouse* and *game* CAMS needs to perform more steps to find good solutions.

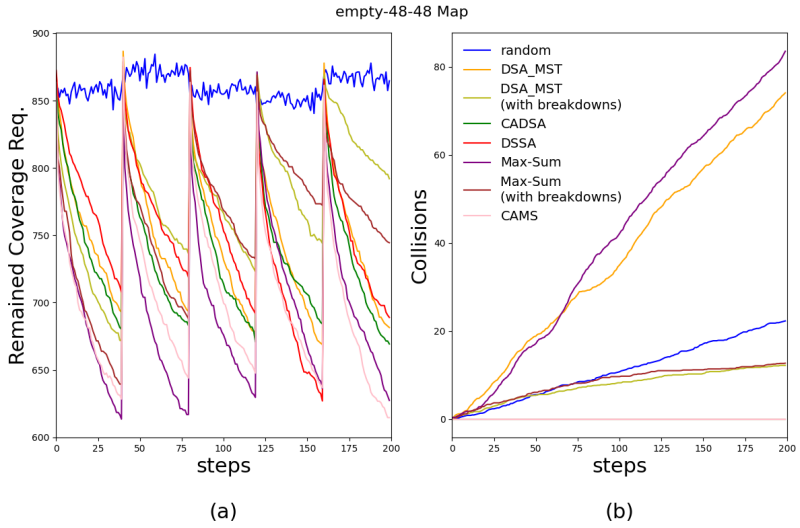


Figure 14: *empty* environment with *dynamic* targets

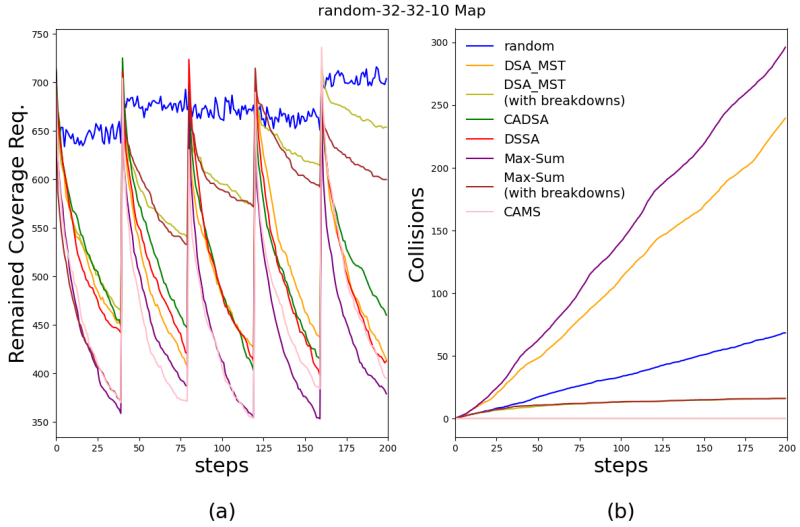


Figure 15: *random* environment with *dynamic* targets

In terms of collisions, we see a similar pattern to the experiments in the static scenarios, where there are more collisions in more open-spaced environments. CAMS does not produce any collisions as expected.

### 7.2 Comparing OVP and BUA

In Section 5 we describe the balanced utility adaptation (BUA) method for adjusting the utility of covering sensors to the requirements of the target. Intuitively, the motivation for designing it was to generate a more balanced method for adjusting the utility offered by targets to their neighboring sensors, with respect to their capabilities. In the ordered value propagation (OVP) method, which was proposed for this purpose in (Yedidsion et al., 2014), all the required reduction was applied to

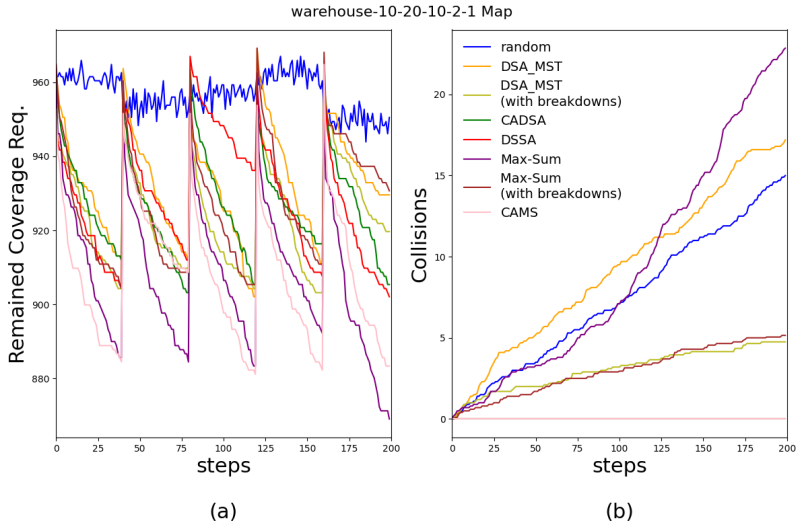


Figure 16: *warehouse* environment with *dynamic* targets

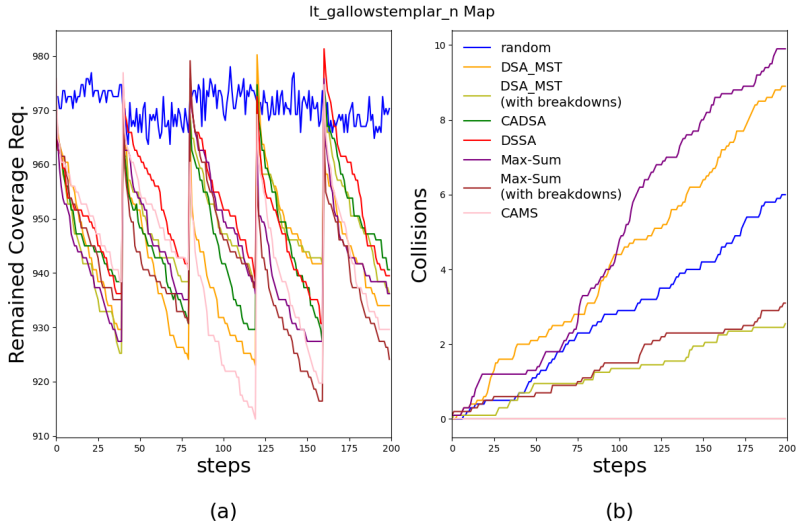


Figure 17: *game* environment with *dynamic* targets

a single sensor. In more detail, each target (function-node) orders its neighboring sensors (variable-nodes) according to the number of constraints they were involved in. The number of constraints is revealed to the targets as part of the FMR procedure. A target assigns utility one by one to the sensors, while the last agent receives the remainder. On the contrary, in BUA, the remainder is evenly distributed across all agents. We note that FMR is used for symmetry breaking (as proposed in (Yedidsion et al., 2014)) and the utility adjustment is required for FMR. Both methods serve this purpose.

Figure 18 presents a comparison between two CAMS versions, one using OVP and the other using BUA. It is clear from the results that our estimations were correct. It is apparent that when using OVP the algorithm performs much more erratically and oscillates, compared to the version that uses BUA. In addition, the version that uses BUA significantly outperforms the version that

uses OVP. The only environment in which the OVP version produces results with similar quality is the game environment. Figure 19 presents similar results in the dynamic settings.

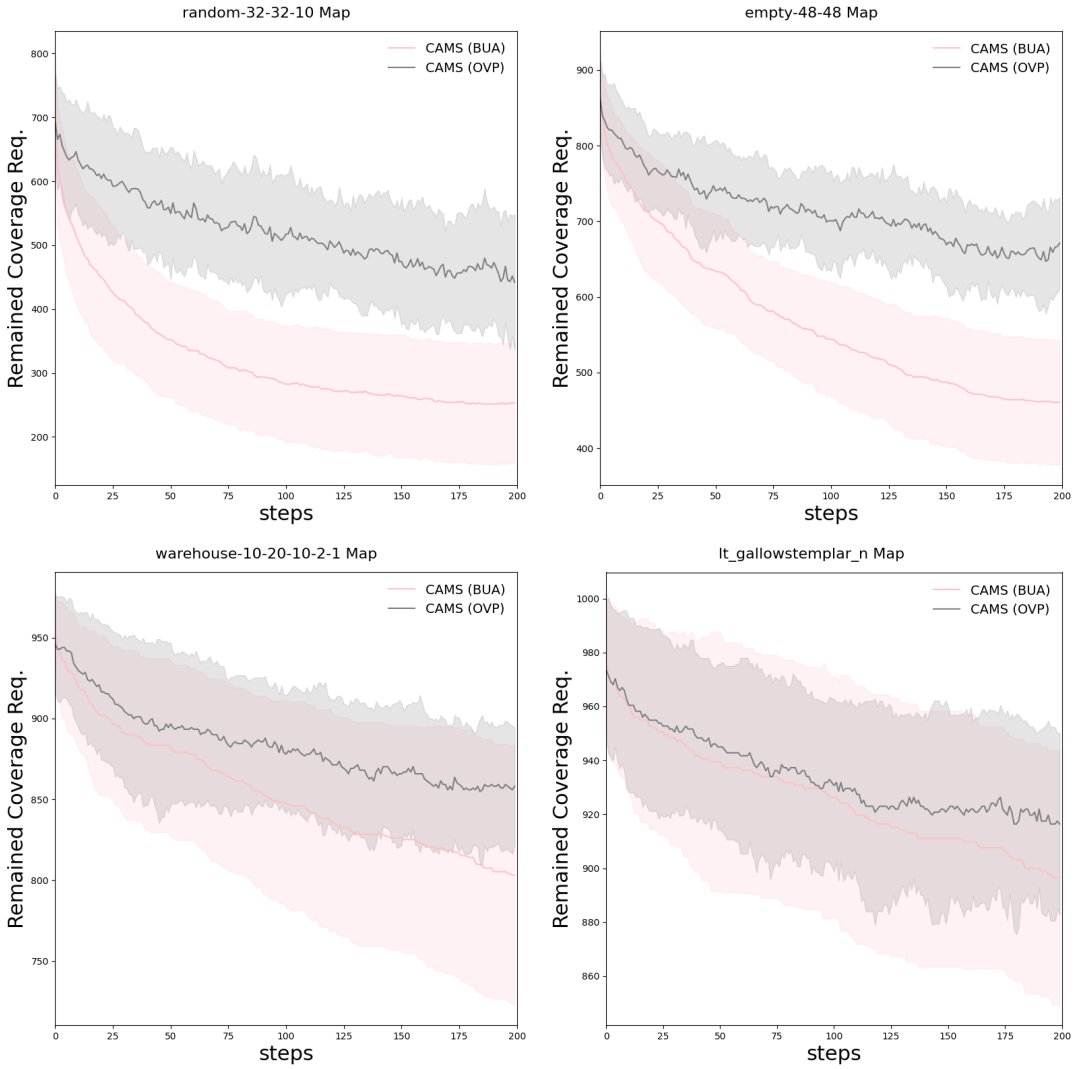


Figure 18: From top to bottom, from left to right: *random*, *empty*, *warehouse* and *game* environments with *static* targets

### 7.3 Evaluation in a Physical Robotic System

In the next set of experiments our goal was to examine the practicality of CAMS in a physical multi-robot system, and specifically to evaluate the delay caused by collisions between robots in such a realistic setting. Thus, we have fully implemented CAMS and Max-sum_MST on a mobile sensing team composed of three Hamster robots (Cogniteam, 2020) and two targets, placed in a $4 \times 4$ grid, where the size of each vertex of the grid was one square meter. The targets' ER was set to 60, and they were placed randomly in non-adjacent vertices of the grid. The sensors' credibility and the targets' requirements were identical to the software simulation experiments. The number of steps

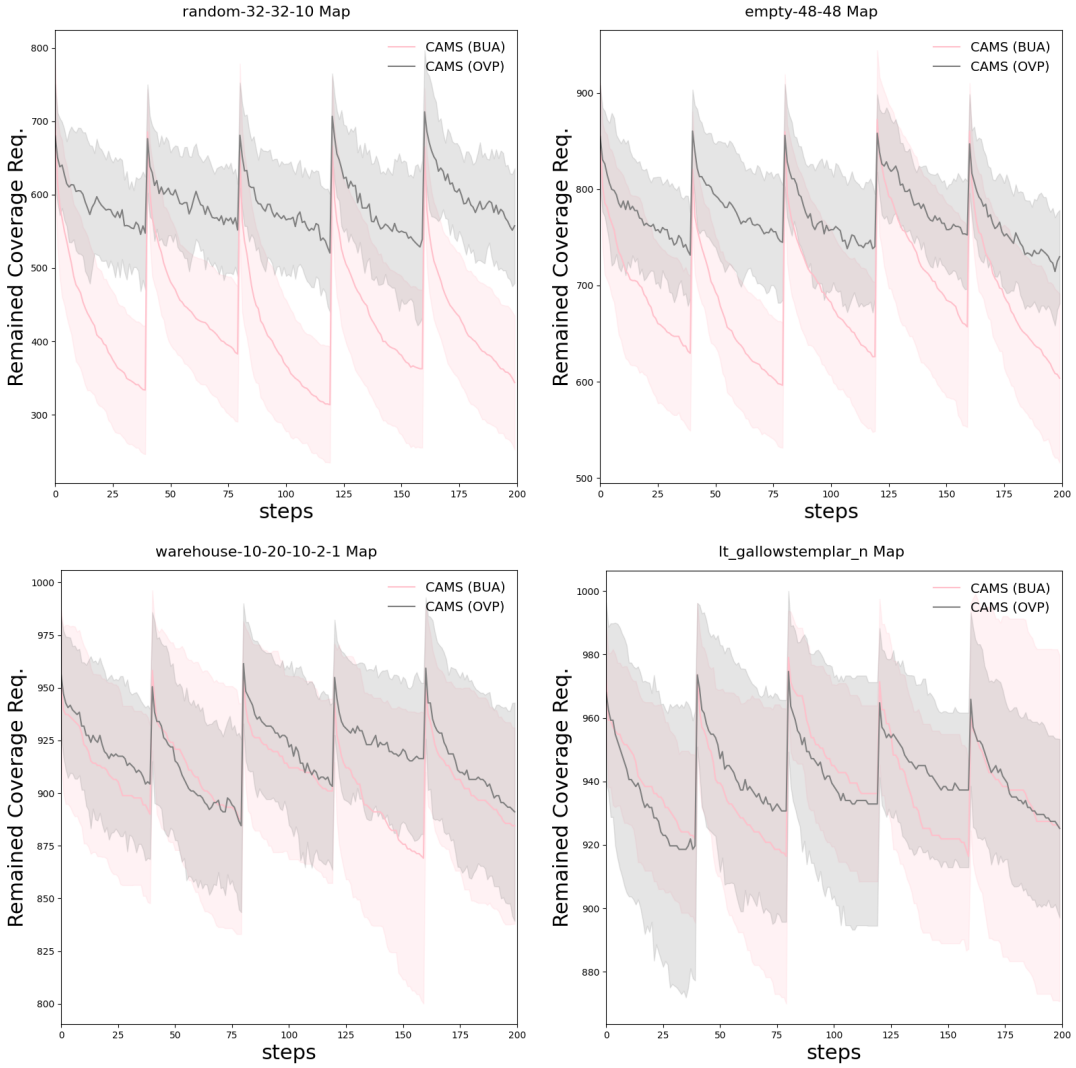


Figure 19: From top to bottom, from left to right: *random*, *empty*, *warehouse* and *game* environments with *dynamic* targets

performed was 10. In each of them, the robots constructed a factor graph in which each of the robots performed the role of its own variable-node. The location and target function-nodes were performed by the nearest robot (qualities were broken by index). Thus, the implementation was completely distributed. The robots performed 30 iterations of Max-sum in each step, before selecting locations, moving, and beginning the next step. When a collision occurred, the robots had to sort out the conflict and continue to the selected location. This resulted in a delay, and thus, we report the time for arriving to the coverage solution. 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. The static obstacles were avoided by using a *move_base* ROS package.

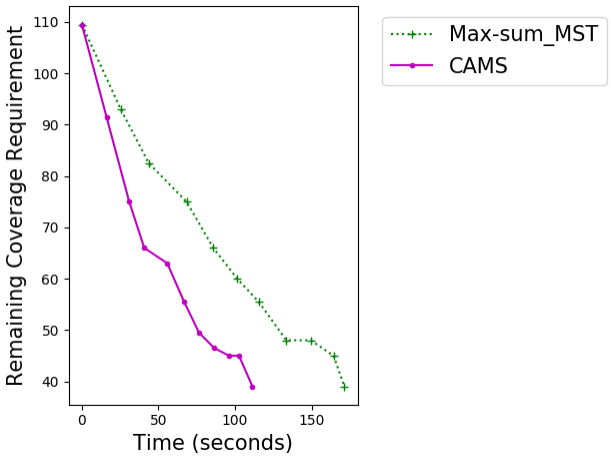


Figure 20: Remaining coverage as a function of time

Figure 20 presents the remaining coverage requirement as a function of the experiment execution time. Both algorithms produced the same level of coverage after completing 10 steps. However, CAMS reached this coverage state faster.

## 8. Conclusion

An important feature of applications that include mobile sensors, is that they should avoid collisions while optimizing coverage. CAMS achieves this challenging combination by adding to the factor-graph representation of the problem, hard constraint function-nodes, representing the locations that mobile sensors may choose to move to. In contrast to what one might expect, although this addition resulted in a much denser factor-graph including many more cycles, it did not prevent the algorithm from converging. Our theoretical analysis gave some insight into this phenomenon. We proved that in Max-sum_MST, the target representing function-nodes sends consistent messages throughout the algorithm run. This result explains the fast convergence of Max-sum_MST, in contrast to the behavior of standard Max-sum when solving other benchmark problems. We proved that when considering only location representing function-nodes, on tree-structured graphs and single-cycle graphs the algorithm is guaranteed to converge to the optimal, collision-free solution. Moreover, adding a target representing function-nodes to these graphs does not change the convergence properties. Moreover, we prove that in every iteration of CAMS and for any graph structure, if the Max-sum algorithm converges, it is to a collision-free solution. Our empirical results revealed that in all cases examined, the algorithm indeed converges. In addition, it demonstrated that the desired properties of CAMS are maintained when the problem scales and in the presence of dynamic events. Most importantly, the advantage of Max-sum_MST over DSA_MST is maintained in the collision-avoiding versions of these algorithms.

## Acknowledgements

Partial support for this research was provided by the Agricultural, Biological, and Cognitive Robotics Initiative at Ben-Gurion University of the Negev, funded by the Helmsley Charitable Trust. Additional support was received from the Marcus Endowment Fund at Ben-Gurion University of the Negev. The research also received funding from ISF grant No. 1070/20.

## Appendix A. Standard Deviation

| Algorithms: | | MS-MST | | MS-MST breakdowns | | CAMS | |
|---|---|---|---|---|---|---|---|
| | | mean | std | mean | std | mean | std |
| static targets | *random* | **261** | 99.69 | 340 | 84.85 | 275 | 93.14 |
| | *empty* | 530 | 75.76 | 580 | 77.46 | **524** | 75.79 |
| | *warehouse* | 831 | 59.15 | 832 | 57.76 | **824** | 58.17 |
| | *game* | 877 | 57.02 | 882 | 41.42 | **872** | 45.78 |
| dynamic targets | *random* | **389** | 91.76 | 668 | 107.03 | 398 | 92.07 |
| | *empty* | **667** | 101.44 | 801 | 81.60 | **677** | 82.29 |
| | *warehouse* | **893** | 53.02 | 901 | 33.75 | 898 | 39.45 |
| | *game* | **928** | 34.87 | 953 | 34.22 | 955 | 28.90 |

Table 2: Comparison between the Max-Sum algorithms and CAMS. The table presents the mean and the standard deviation of the *remained coverage requirement* following the last iteration.

| Algorithms: | | DSA_MST | | DSA_MST breakdowns | | CADSA | | DSSA | | CAMS | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | mean | std | mean | std | mean | std | mean | std | mean | std |
| static targets | *random* | 290 | 75.13 | 372 | 74.49 | 296 | 82.15 | 305 | 57.33 | **262** | 73.63 |
| | *empty* | 495 | 92.28 | 554 | 79.54 | 486 | 107.15 | 473 | 88.77 | **441** | 103.33 |
| | *warehouse* | 809 | 50.97 | 838 | 30.54 | 806 | 56.78 | 808 | 47.95 | **805** | 63.73 |
| | *game* | 903 | 45.83 | 906 | 55.16 | 906 | 50.11 | 908 | 44.15 | **893** | 48.84 |
| dynamic targets | *random* | 414 | 88.00 | 654 | 106.90 | 460 | 82.06 | 412 | 86.17 | **395** | 127.32 |
| | *empty* | 681 | 79.93 | 792 | 79.67 | 669 | 56.30 | 689 | 60.37 | **614** | 66.71 |
| | *warehouse* | 929 | 25.66 | 919 | 48.34 | 905 | 38.80 | 902 | 32.98 | **883** | 60.69 |
| | *game* | 934 | 34.08 | 936 | 46.62 | 940 | 42.94 | 939 | 36.07 | **929** | 39.11 |

Table 3: Comparison between the local search algorithms and CAMS. The table presents the mean and the standard deviation of the *remained coverage requirement* following the last iteration.

## References

Atzmon, D., Stern, R., Felner, A., Wagner, G., Barták, R., & Zhou, N. (2020). Robust multi-agent path finding and executing. *Journal of Artificial Intelligence Research (JAIR)*, *67*, 549–579.

Chen, Z., Deng, Y., Wu, T., & He, Z. (2018). 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.

Chen, Z., Jiang, X., Deng, Y., Chen, D., & He, Z. (2019). A generic approach to accelerating belief propagation based incomplete algorithms for dcops via a branch-and-bound technique. In *The Thirty-Third Conference on Artificial Intelligence, (AAAI), Honolulu, Hawaii, USA*, pp. 6038–6045.

Cogniteam (2020). *Hamster V7 Smart ROS Autonomous Ground Vehicles for Industry and Academic R & D*.

Cohen, E., Lev, O., & Zivan, R. (2023). Separate but equal: Equality in belief propagation for single cycle graphs. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*, pp. 3924–3931.

Cohen, L., Galiki, R., & Zivan, R. (2020). Governing convergence of max-sum on dcops through damping and splitting. *Artif. Intell.*, *279*.

DeCastro, J. A., Alonso-Mora, J., Raman, V., Rus, D., & Kress-Gazit, H. (2018). Collision-free reactive mission and motion planning for multi-robot systems. In *Robotics research*, pp. 459–476. Springer.

Deng, Y., & An, B. (2020). 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)*, pp. 31–38.

Farinelli, A., Rogers, A., Petcu, A., & Jennings, N. R. (2008). 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)*, pp. 639–646.

Farinelli, A., Rogers, A., & Jennings, N. R. (2014). Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, *28*(3), 337–380.

Forney, G. D., Kschischang, F. R., Marcus, B., & Tuncel, S. (2001). Iterative decoding of tail-biting trellises and connections with symbolic dynamics. In *Codes, Systems, and Graphical Models*, pp. 239–264. Springer.

Hirayama, K., Miyake, K., Shiota, T., & Okimoto, T. (2019). Dssa+: Distributed collision avoidance algorithm in an environment where both course and speed changes are allowed. *The International Journal on Marine Navigation and Safety of Sea Transportation*, *13*.

Hoang, K. D., Fioretto, F., Hou, P., Yeoh, W., Yokoo, M., & Zivan, R. (2022). Proactive dynamic distributed constraint optimization problems. *J. Artif. Intell. Res.*, *74*, 179–225.

Hoang, K. D., Yeoh, W., Yokoo, M., & Rabinovich, Z. (2020). New algorithms for continuous distributed constraint optimization problems. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, pp. 502–510.

Hoy, M., Matveev, A. S., & Savkin, A. V. (2015). Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. *Robotica*, *33*(3), 463–497.

Jain, M., Taylor, M. E., Yokoo, M., & Tambe, M. (2009). 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)*, pp. 181–186.

Khan, M. M., Tran-Thanh, L., Ramchurn, S. D., & Jennings, N. R. (2018). Speeding up gdl-based message passing algorithms for large-scale dcops. *Comput. J.*, *61*(11), 1639–1666.

Kim, Y., & Lesser, V. R. (2013). Improved max-sum algorithm for dcop with n-ary constraints. In *Proceeding of the 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

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

Macarthur, K. S., Stranders, R., Ramchurn, S. D., & Jennings, N. R. (2011). 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)*.

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

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Pertzovskiy, A., Zivan, R., & Agmon, N. (2023). CAMS: collision avoiding max-sum for mobile sensor teams. In Agmon, N., An, B., Ricci, A., & Yeoh, W. (Eds.), *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023*, pp. 104–112. ACM.

Pianpak, P., Son, T. C., Toups, Z. O., & Yeoh, W. (2019). A distributed solver for multi-agent path finding problems. In *Proceedings of the First International Conference on Distributed Artificial Intelligence*, pp. 1–7.

Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., Rodriguez-Aguilar, J. A., & Tambe, M. (2013). Engineering the decentralized coordination of uavs with limited communication range. *Advances in Artificial Intelligence*, *1*, 199–208.

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

Rogers, A., Farinelli, A., Stranders, R., & Jennings, N. R. (2011). Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, *1*.

Rust, P., Picard, G., & Ramparany, F. (2016). 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)*, pp. 468–474.

Salzman, O., & Stern, R. (2020). Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pp. 1711–1715.

Sarker, A., Choudhury, M., & Khan, M. M. (2021). A local search based approach to solve continuous dcops. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1127–1135.

Serra-Gómez, Á., Brito, B., Zhu, H., Chung, J. J., & Alonso-Mora, J. (2020). With whom to communicate: learning efficient communication for multi-robot collision avoidance. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11770–11776. IEEE.

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence journal (AIJ)*, *219*, 40–66.

Stern, R., Sturtevant, N. R., Felner, A., Koenig, S., Ma, H., Walker, T. T., Li, J., Atzmon, D., Cohen, L., Kumar, T. S., et al. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*.

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

Sturtevant, N. R. (2012). Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(2), 144–148.

Tabasso, C., Cichella, V., Mehdi, S. B., Marinho, T., & Hovakimyan, N. (2021). Time coordination and collision avoidance using leader-follower strategies in multi-vehicle missions. *Robotics*, *10*(1), 34.

Taylor, M. E., Jain, M., Jin, Y., Yokoo, M., & Tambe, M. (2010). When should there be a "me" in "team"?: distributed multi-agent optimization under uncertainty. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 109–116.

Voice, T., Stranders, R., Rogers, A., & Jennings, N. R. (2010). A hybrid continuous max-sum algorithm for decentralised coordination. In *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, pp. 61–66.

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

Weiss, Y. (2000). Correctness of local probability propagation in graphical models with loops. *Neural Computation*, *12*(1), 1–41.

Yedidsion, H., Zivan, R., & Farinelli, A. (2014). Explorative max-sum for teams of mobile sensing agents. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS*, pp. 549–556.

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

Zhang, W., Xing, Z., Wang, G., & Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: properties, comparishon and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, *161:1-2*, 55–88.

Zivan, R., Lev, O., & Galiki, R. (2020). 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)*, pp. 7333–7340.

Zivan, R., Parash, T., Cohen, L., Peled, H., & Okamoto, S. (2017). 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.

Zivan, R., Yedidsion, H., Okamoto, S., Glinton, R., & Sycara, K. P. (2015). Distributed constraint optimization for teams of mobile sensing agents. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, *29*(3), 495–536.