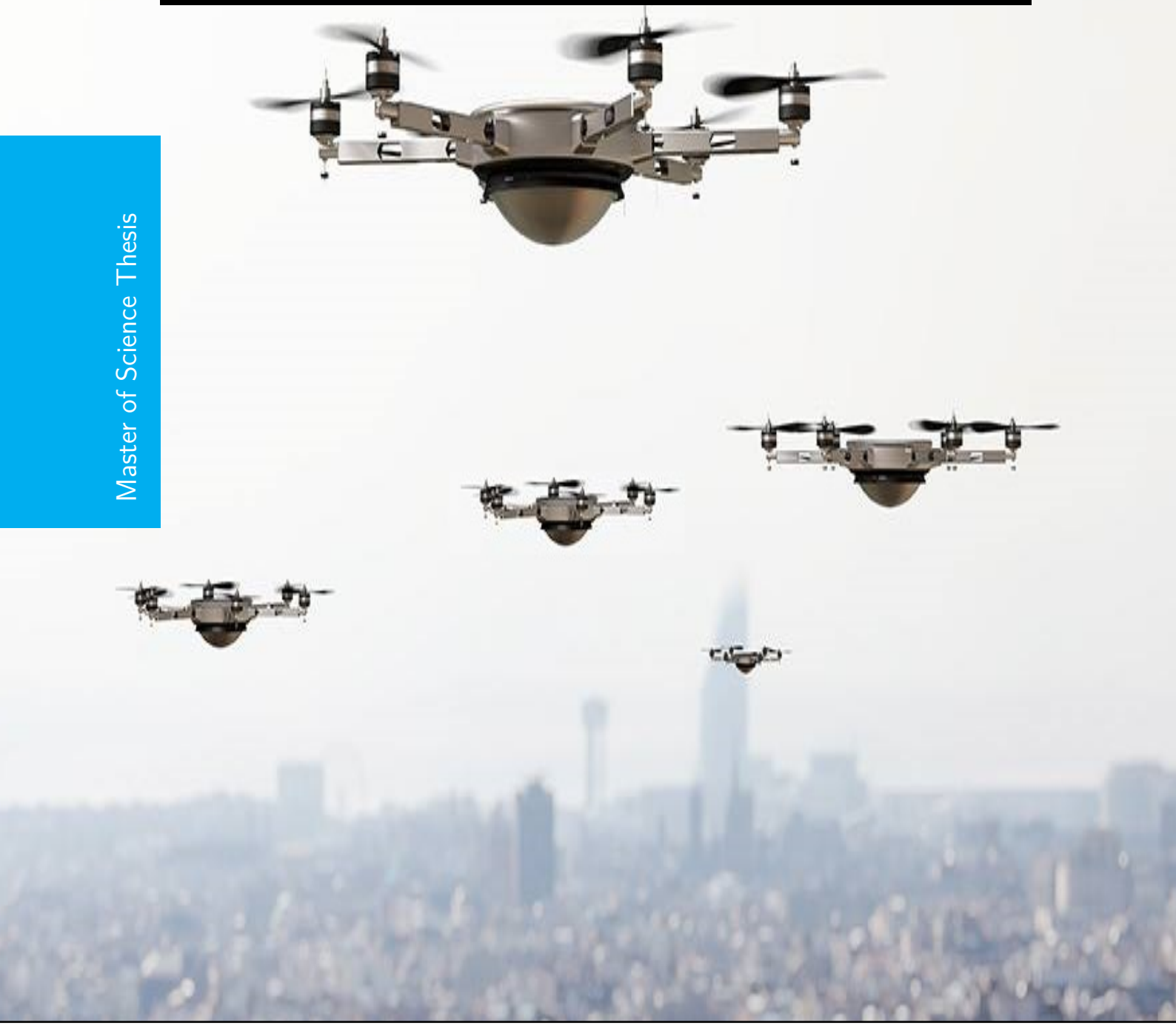
Decentralized coordination for area surveillance purposes

Kyriakos Demetriou

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Decentralized coordination for area surveillance purposes

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Kyriakos Demetriou

March 29, 2019

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Decentralized coordination for area surveillance purposes

by

Kyriakos Demetriou

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: March 29, 2019

Supervisor(s):

dr.ir. Joris Sijs

ir. Jeroen Fransman

Reader(s):

prof.dr.ir. Bart De Schutter

dr. Riccardo Ferrari

Abstract

The area surveillance problem is the problem of surveying a known or an unknown area with the main purpose of detecting objects. This thesis will tackle the problem of how to employ a group of mobile-sensors for surveying an unobstructed area in an optimal manner. The mobile-sensors should make use of their onboard computers and they should iteratively compute their waypoints until they successfully surveyed the area.
To solve the area surveillance problem in an optimal manner, the mobile-sensors should be able to coordinate their actions. The Distributed Constraint Optimization Problem (DCOP) framework will be employed to define the area surveillance problem. In the DCOP, the mobile-sensors can define their optimal position for the next discrete time step by communicating with the other mobile-sensors that participate in the problem.
For solving this DCOP, the Distributed Pseudo-tree Optimization Procedure (DPOP) will be utilized. The DPOP is a complete solver that can find the optimal solution of a DCOP in a decentralized manner. However, the main limitation of DPOP is that the size of the largest message that the mobile-sensors will have to exchange is space-exponential in the induced width of the pseudo-tree. Considering that the mobile-sensors have to use their onboard computers for solving the problem, exchanging huge size of messages is not desired. To overcome this limitation of DPOP, a new extension, known as the MST-DPOP will be presented in this Thesis.
The MST-DPOP makes use of the Maximum Spanning Tree (MST) algorithm to reduce the size of the largest message that the mobile-sensors have to exchange. Employing MST along with DPOP can bound the size of the largest message and the required computations for constructing the utility messages. However, the new extension cannot guarantee that its solution will be the optimal. The experimental results shows that the MST-DPOP is able to define a solution with an average error less than 2%. Moreover, the MST-DPOP requires on average around 1 discrete time step more than the DPOP to solve the area surveillance problem. Consequently, given that the MST-DPOP overcomes the high memory requirements of DPOP, it is preferred for solving the area surveillance problem.

Table of Contents

List of Figures

List of Tables

List of Acronyms

MMRS	Multi Mobile Robot System
FoV	Field of View
UAV	Unmanned Aerial Vehicles
DoF	Degrees of Freedom
COP	Constraint Optimization Problem
DCOP	Distributed Constraint Optimization Problem
DPOP	Distributed Pseudo-tree Optimization
ADOPT	Asynchronous Distributed OPTimization
DFS	Depth-First Search
MGM	Maximum Gain Message
DSA	Distributed Stochastic Algorithm
BFS	Breadth-First Search
MST	Maximum Spanning Tree

Acknowledgements

I would first like to express my sincere gratitude to my supervisors dr.ir. Joris Sijs and ir. Jeroen Fransman for the continuous support of my Master thesis research, for their patience, motivation, enthusiasm, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having better advisors and mentors for my Mater thesis study.

Besides my advisors, I wish to thank the members of my dissertation committee prof.dr.ir. Bart De Schutter and dr. Riccardo Ferrari for generously offering their time, support and good will throughout the preparation and review of this document.

I want to express my very profound appreciation to my wonderful parents, my brothers and to my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this Master thesis. This accomplishment would not have been possible without them. Thank you.

Finally, I would like to thank my friend Gavriella for being so supportive while I was working on my thesis. Thanks for your emotional support for being so understanding and for being such a great friend.

Delft, University of Technology
March 29, 2019

Kyriakos Demetriou

Chapter 1

Introduction

Area surveillance is the problem of surveying a known or an unknown area with the main purpose of detecting objects. Depending on the problem, the objects that have to be discovered may differ in type. For example, after a physical disaster, like an earthquake or a flood, a rescue and resource team is searching for survivors. Moreover, under a war situation, the military wants to protect its base station from intruders.
Currently, this operation is executed by humans. Humans can explore the environment and detect the corresponding objects. However, this operation might be time-consuming and the result may not be sufficient. That is, the objects may not be detected. Furthermore, it could be that this operation is of high risk for humans. Consequently, an alternative solution is needed, such as the use of mobile-sensors.
Mobile-sensors are robots equipped with sensors, as can be seen in Figure 1-1. They are applicable to a variety of real-world applications due to their ease of use and their low cost. By employing mobile-sensors for the area surveillance problem, the required surveillance time can be reduced and the performance can be enhanced. Yet, a single mobile-sensor will not be able to reduce the surveillance time and to improve the quality of object detection, especially if the environment that has to be explored is large. As a result, several mobile-sensors have to be used to perform the surveillance in parallel, where their actions must be coordinated. These systems are known as Multi Mobile Robot Systems (MMRSs). MMRSs consist of several intelligent robots, where each robot has the ability to perform certain tasks given its observations of the environment [3]. This thesis will make use of the MMRS to solve the area surveillance problem.

1-1 Area surveillance

During the last few decades, MMRSs have been used in a variety of applications, such as maximal area coverage [4, 5, 6, 7], resource allocation in disaster scenarios [8]. Thanks to their high success on these problems, MMRSs can be considered as suitable systems that can be utilized for performing area surveillance. In the area surveillance problem, the mobile-sensors

Figure 1-1: Example of mobile-sensors.

aim to survey an area as fast as possible. In this thesis, the focus is given only on how to survey a given area and the detection of objects will not be further examined. Consequently, the problem of employing mobile-sensors to survey a given area in an optimal manner will be studied.
The area surveillance is assumed to take place in a $2D$ area, which is discretized into multiple grid points. In order to successfully survey the given area, the mobile-sensors have to observe every grid point that exists in the environment. In addition, depending on the type of the sensor that the mobile-sensors carry and their current location, there are a certain number of grid points that they can simultaneously perceive. Consequently, to scan every grid point in the environment and solve the area surveillance problem, the mobile-sensors have to adjust their positions. However, to achieve this in an optimal manner, the mobile-sensors have to cooperate with each other before adjusting their positions, otherwise, they may end up surveiling the same grid points of the environment. This can be modeled as a constraint optimization problem, where the solution of this optimization problem can be either a trajectory or a single target position for every mobile sensor.

Trajectory vs target position

A trajectory is a set of target positions for every mobile-sensor that participate in the problem. In the case of the area surveillance problem, a trajectory specifies a sequence of positions that every mobile-sensor has to follow, such that they accomplish to survey every grid point of the given area. The trajectory for a mobile-sensor is computed once, and then the mobile-sensor has to follow the sequence of positions. Alternatively, a waypoint defines the position that every mobile-sensor has to reach, assuming that it can take only one action. The mobile-sensors iteratively determine new target positions until they successfully inspect every grid point. Both approaches, trajectory or waypoint, can result in covering the whole area. Nevertheless, both of them have their advantages and disadvantages.
The trajectory approach will be computationally expensive to determine the trajectory for all of the mobile-sensors that participate in the area surveillance problem. In addition, since in the MMRSs the mobile-sensors are equipped with onboard computers that have limited com-

putational resources, the onboard computers will not be able to perform such computations for problems with large number of mobile-sensors. Hence, a computer is needed to compute the trajectory for every mobile-sensor. Furthermore, if a failure occurs during the surveillance, that is, a failure of a mobile-sensor or something changing in the area under surveillance, for example a new object appeared, new trajectories need to be calculated again. That makes the trajectory approach less suitable to dynamic behaviors, especially when a large number of mobile-sensors is used.
On the contrary, the waypoint approach will result in surveying the given area step-by-step. That is, at every step, the mobile-sensors have to define their optimum position, such that they will observe the maximum number of unsurveilled grid points. This procedure is known as a one-step-ahead optimization. In one-step-ahead optimization, the required computational power is less than the trajectory approach. This allows the mobile-sensors to use their onboard computers for solving the one-step-ahead optimization. Therefore, this makes the one-step-ahead approach more suitable for dynamic environments, since it can be run during the surveillance. Nevertheless, one disadvantage for the one-step-ahead optimization is that the mobile-sensors might require to take more actions to successfully survey the given area since during the optimization procedure the mobile-sensors define their optimum position considering only the next step.
Utilizing the one-step-ahead optimization, the mobile-sensors will be able to run all the required computations on their onboard computers. Therefore, the waypoint approach is preferred over the trajectory approach. Additionally, the available computation resources of the onboard computers make centralized computation infeasible. Therefore, a decentralized approach will be derived for solving the area surveillance problem.
The rest of this thesis will tackle the problem of solving the area surveillance problem in a decentralized manner using only the onboard computers of the mobile-sensors. In addition, the one-step-ahead optimization will be utilized for calculating the waypoints of the mobile-sensors at every discrete time step.

1-2 Problem formulation

Regarding the area surveillance problem, the mobile-sensors have to cooperate and coordinate their actions to survey the given area with an optimal manner. This implies that the mobile-sensors have to communicate with each other before selecting their next waypoint. Before getting into more details on how to formulate the area surveillance problem, the following assumptions for the problem are defined.

- N mobile-sensors are utilized, where N should be larger than 1.
- The area surveillance problem is a discrete time problem, where an iteration implies a single discrete time step, k.
- Every mobile-sensor has computational resources to run onboard algorithms.
- Each mobile-sensor has a specific sensing range. That is, a mobile-sensor can observe the area that is within its sensing range.
- Each mobile-sensor has a specific mobility range. Thus, a mobile-sensor can move within this range in a single discrete time step.

- The current position of the mobile-sensor is always known by the mobile-sensor.
- Control of the position of the mobile-sensor is possible by assigning a target position.
- The mobile-sensors have to survey an unobstructed known area. This is a $2D$ rectangle area.

Furthermore, there are several concepts that have to be introduced. First, the environment that represents the area under surveillance has to be modeled. Moreover, mobile-sensors have constraints based on their mobility and sensing abilities. Therefore, a description of mobility and sensing constraint will be determined. Additionally, the model that describes how a mobile-sensor can interact with the environment, known as interaction model, will be defined. In addition, the communication structure has to be specified. Therefore, a definition of the utility constraint will take place. Finally, since a one-step ahead optimization will be used, the goal of the mobile-sensors for every discrete time step will be defined. Table 1-1 shows an overview of the notations that will be used.

Table 1-1: Formal notation for area surveillance problem.

Notation	Description
N	Number of mobile-sensors.
M	Model of the environment.
p_i	Probability value of grid point i.
d_κ	Location of mobile-sensor κ.
θ_κ	Heading of mobile-sensor κ.
D_{R_κ}	Sensing range domain of mobile-sensor κ.
D_{M_κ}	Mobility domain of mobile-sensor κ.
D_{A_κ}	Active domain of mobile-sensor κ.
$D_{S_{\kappa\rho}}$	Shared utility domain between mobile-sensor κ and ρ.
$\oplus$	Minkowski sum.

1-2-1 Environment model

The map of the environment represents the area that the mobile-sensors should survey. In this problem, the area is assumed to be rectangular. Furthermore, using the cell decomposition approach, the map is decomposed into a finite number of grid points. Hence, the map consists of several grid points, where each grid point has its own property. In the area surveillance problem, this property is a probability value that expresses the likelihood that an object exists in the corresponding grid point. The map of the environment has the following characteristics:

- Center location for each grid point i with respect to a global coordinate system $d_i = (x_i, y_i)$, where $d_i \in R^2$.
- A probability that a grid point contains an object, $p_i(k)$, where $p_i(k) \in [0, 1]$.

Consequently, the model of the environment can be described by the following:

$$M := \{\bar{x}_1, \bar{x}_2, ..., \bar{x}_n\} \tag{1-1}$$

where $\bar{x}_i = [d_i, p_i(k)]$, and n is the total number of grid points.

1-2-2 Sensing constraint

The sensing constraint depends on the type of the sensor that will be utilized. There are several sensors that can be employed for surveying a given area. Some examples are lidar, camera, ultra sound. Cameras have been used in a variety of applications for detecting objects successfully [9, 10]. For that reason, the mobile-sensors are equipped with a camera that has 360^o Field of View (FoV). The sensing constraint depends on the camera's range r_κ and the location d_κ of the camera. The sensing range domain is defined as the set of grid points that the sensor of the mobile-sensor κ can survey if the sensor is at location d_κ. It is defined as a circle, where the center of the circle is the location of the mobile-sensor and its radius is the camera's range. Therefore, by using the location of each grid point $d_i \in M$, the camera's range r_κ and the current location of the mobile-sensor d_κ, the sensing range domain $D_{R_\kappa} \subseteq M$ can be derived. A grid point belongs in the D_{R_κ} if it is within the range r_κ of the mobile-sensor κ. Consequently, the domain D_{R_κ} is defined as Equation 1-2.

$$D_{R_\kappa} := \{\bar{x}_i \in M \mid |d_\kappa - d_i|_2 \leq r_\kappa\} \tag{1-2}$$

A $2D$ example of the sensing range domain can be seen in Figure 1-2.

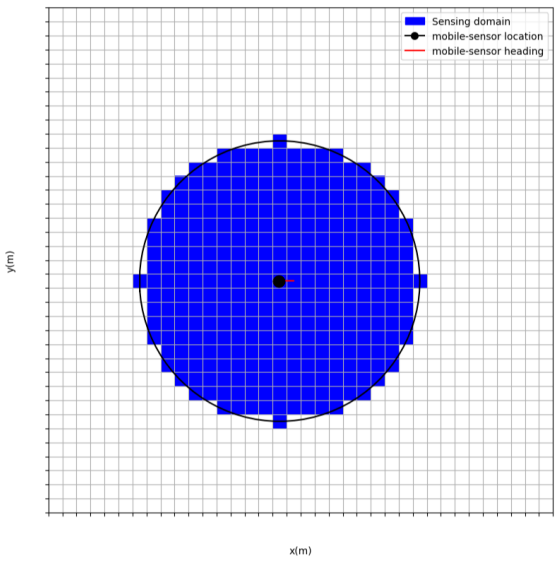


Figure 1-2: Example of sensing range domain.

1-2-3 Mobility constraint

The mobility constraint depends on the type of the mobile robot that will be used. Even though there are numerous types of robots that have the ability to perform area surveillance, the Unmanned Aerial Vehicles (UAVs) are best suited for this problem. This is because UAVs are agile and able to fly on high speeds. Furthermore, UAVs are able to overcome obstacles that ground robots cannot.

UAVs have in total six Degrees of Freedom (DoF). They have the ability to move in three translation axes (x, y, z), and also to rotate around these axes (roll, pitch, yaw). Though, for the area surveillance problem, it is reasonable to ignore some of these DoF. More specifically, the pitch and the roll angles are not of high interest. They will not have a big effect on the sensor's FoV if they are kept close to zero. This is because of the fact that the camera can be attached on the UAV such that it always looks downwards. Moreover, the z dimension can be ignored too, by assuming that the UAVs fly at the same height during the surveillance. The height can be selected such that is the optimum for detecting objects using the camera. Consequently, the pose of the UAV will depend only to the x, y, and yaw angle. This is the same with a unicycle mobile robot where it has 3DoF, Figure 1-3.

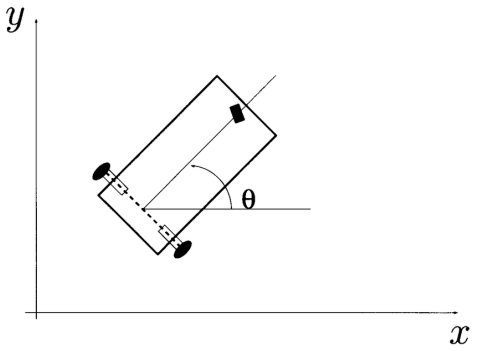


Figure 1-3: Example of a unicycle mobile robot.

Therefore, the discrete model of the unicycle mobile robot, Equation 1-3, will be utilized as the model for the UAV.

$$\begin{aligned} x(k+1) &= x(k) + u\,cos(\theta(k)) \\ y(k+1) &= y(k) + u\,sin(\theta(k)) \\ \theta(k+1) &= \theta(k) + \delta\theta \end{aligned} \tag{1-3}$$

where θ is the yaw angle of the UAV and $\delta\theta$ is the angular velocity.
The mobility domain of the UAV can be derived using the discrete model of Equation 1-3. The mobility domain defines all the possible positions in the environment that the mobile-sensor can reach in a single discrete time step. The mobility domain will be derived based on the speed and the heading of the mobile sensor.
Every mobile-sensor is limited by its turn rate and its maximum speed. These limits are known as maximum turn rate θ_{max} and maximum speed u_{max}. To be more precise, using the maximum turn rate and the current heading of the mobile sensor, a maximum range for the heading can be obtained. This range is defined using Equation 1-4.

$$D_{\theta_\kappa} = [\theta_{min_\kappa}(k), \theta_{max_\kappa}(k)] = [\theta_\kappa(k) - \theta_{max}, \theta_\kappa(k) + \theta_{max}] \tag{1-4}$$

Additionally, by utilizing the current speed and the maximum speed, a range for the speed of the mobile robot (D_{u_κ}) can be derived in a similar manner. This range is defined using Equation 1-5.

$$D_{u_\kappa} = [u_{min_\kappa}(k), u_{max_\kappa}(k)] = [0, u_{max}] \tag{1-5}$$

Finally, by combining the heading range and the speed range, the mobility range can be obtained. The mobility range forms a sector, where its radius is equal with the $u_{max_\kappa}(k)$, and its central angle is equal to $2 \times \theta_{max}$. Figure 1-4 shows an example of a mobility range.

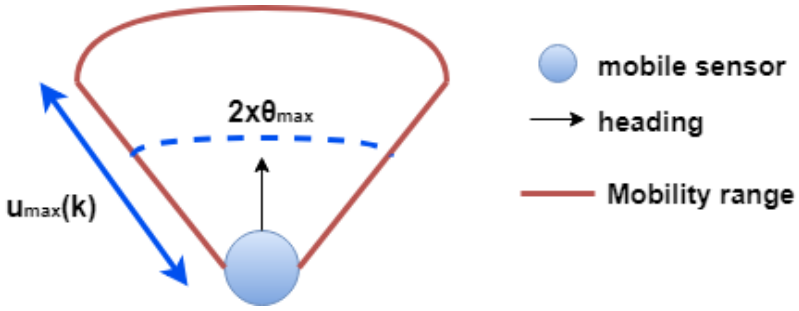


Figure 1-4: Example of the mobility range.

Therefore, the mobility domain D_{M_κ} contains every grid point of the environment that falls within the mobility range. These grid points represent the potential positions of the mobile-sensor at the next discrete time step. To define whether a grid point belongs in the mobility domain, its distance $d_{i,\kappa}$ and its angle $\theta_{i,\kappa}$ with respect to the position and heading of the mobile sensor have to be checked. Figure 1-5 gives an example of the distance and the angle of a grid point.

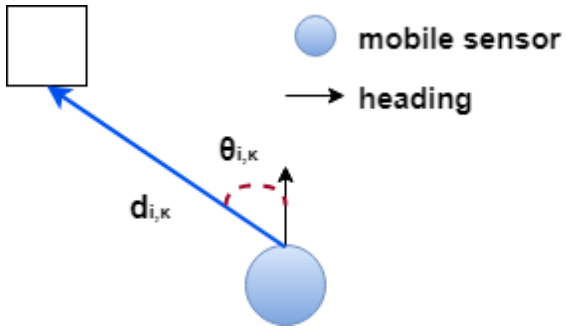


Figure 1-5: The distance ($d_{i,\kappa}$) and the angle ($\theta_{i,\kappa}$) of a grid point i with respect to the mobile-sensor κ.

Finally, the mobility domain D_{M_κ} is defined as Equation 1-6.

$$D_{M_\kappa} := \{\bar{x}_i \in M \mid d_{i,\kappa} \leq u_{max_\kappa}(k)\Delta t, \theta_{i,\kappa} \leq \theta_{max}\Delta t\} \tag{1-6}$$

A $2D$ example of the mobility domain can be seen in Figure 1-6.

1-2-4 Interaction model

As previously mentioned, mobile-sensors aim to observe every grid point that exists in the environment. Since every grid point in the environment has its own probability value, successfully surveying the given area implies to set the probability value of every grid point to zero. An interaction model is used to model the sensing of a grid point. The mobile-sensors

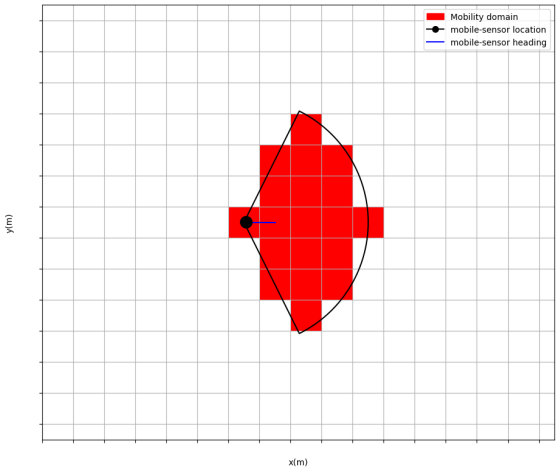


Figure 1-6: Example of a mobility domain.

can interact with the environment using their sensor. For the area surveillance problem, the probability value of a grid point is set to zero if at least one mobile-sensor has the corresponding grid point within its sensing range.
Assuming that a mobile-sensor is at position d_κ, the sensing domain can provide the corresponding grid points that the current mobile-sensor can observe. Therefore, the probability value of these grid points can be set to 0 since the mobile-sensor has just surveyed them. In addition, the probability value of a grid point depends also on its adjacent grid points since it is possible that the objects that exist in the area under surveillance, have the ability to navigate around. Equation 1-7 defines the interaction model between the mobile-sensors and the environment,

$$p_i(k) = \begin{cases} 0 & if\ \bar{x}_i \in \bigcup_{\kappa=1}^{N} D_{S_\kappa} \\ p_i(k-1) + \frac{\sum_{j \in D_n} p_j(k-1)}{|D_n|} & \text{otherwise} \end{cases} \tag{1-7}$$

where D_n consists of the adjacent grid points of the grid point i.

1-2-5 Utility constraint

Equation 1-7 implies that the probability value of a grid point can be set to zero as soon as a single mobile-sensor covers the corresponding grid point. Hence, having two or more mobile-sensors covering the same grid point of the environment is not desired. To improve the performance of the one-step-ahead optimization, two mobile-sensors will have to communicate for computing their waypoint for the next discrete time step if any potential position in their mobility domain result in covering the same grid points.
To determine whether two mobile-sensors have to communicate for defining their waypoint, a new domain named the active domain is defined. The active domain (D_{A_κ}) includes all the grid points that can be covered by the mobile-sensor based on its mobility and sensing domain. Consequently, the active domain takes into consideration the mobility domain and the sensing domain and by combining them together it derives the new domain. The Minkowski sum is

utilized for combining these two domains. Equation 1-8 shows how the Minowski sum is formulated for two domains D_1 and D_2.

$$D := D_1 \oplus D_2 = \{[(x_i + x_j), (y_i + y_j)] \,|\, \forall (x_i, y_i) \in D_1, \forall (x_j, y_j) \in D_2\} \tag{1-8}$$

However, to apply the Minkowski sum to the mobility and sensing domain, an extra step is required. First, the location of the current mobile-sensor should be subtracted from every grid point in the sensing and mobility domain ($D'_{M_\kappa} = \{(x_i - x_\kappa, y_i - y_\kappa) | \forall i \in D_{M_\kappa}\}$, $D'_{R_\kappa} = \{(x_i - x_\kappa, y_i - y_\kappa) | \forall i \in D_{R_\kappa}\}$). Then, the domain D'_{A_κ} for the mobile-sensor κ is defined using the Minkowski sum as shown in Equation 1-9.

$$D'_{A_\kappa} := D'_{M_\kappa} \oplus D'_{R_\kappa} \tag{1-9}$$

Finally, the active domain is defined by adding to every grid point in the domain D'_{A_κ} the location of the mobile-sensor $D_{A_\kappa} = \{(x_i + x_\kappa, y_i + y_\kappa) | \forall (x_i, y_i) \in D'_{A_\kappa}\}$. Figure 1-7 shows the active domain of the corresponding platform. The blue grid points represent the sensing domain and the green grid points the area that the mobile-sensor is able to cover in the next discrete time step.

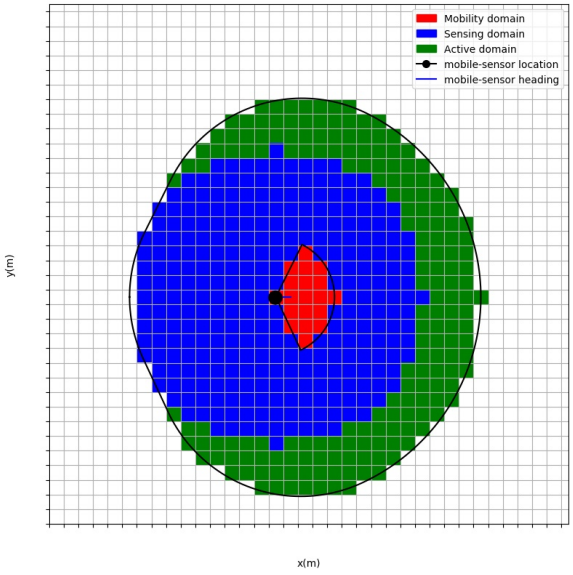


Figure 1-7: Example of active domain where the blue grid points represents the current sensing domain, the red correspond to the mobility domain and the green to the active domain.

Furthermore, if the active domain of two different mobile-sensors include the same grid points of the environment, then these two mobile-sensors have to communicate for solving the problem. In this case, the two mobile-sensors are said to have a utility constraint. To mathematically define whether two mobile-sensors have a utility constraint, Equation 1-10 will be utilized.

$$D_{S_{\kappa\rho}} := D_{A_\kappa} \cap D_{A_\rho} \tag{1-10}$$

The domain $D_{S_{\kappa\rho}}$ is known as shared domain. If the shared domain between mobile-sensor κ and ρ is not empty, then the mobile-sensor κ has a utility constraint with the mobile-sensor ρ. Figure 1-8 shows two mobile-sensors that share a utility constraint.

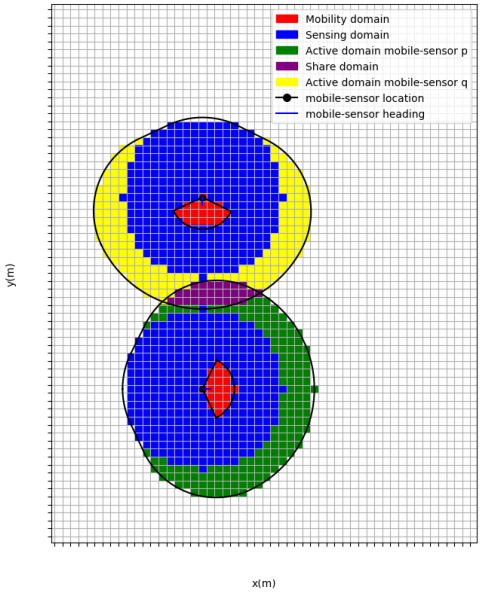


Figure 1-8: Example of the active domain of two different mobile-sensors where the purple grid points show the overlapping area of the two mobile-sensors.

1-2-6 One-step ahead optimization

So far, the interaction model, the model of the environment and the mobility, sensing and utility constraints have been defined. Based on these constraints and given the fact that a one-step-ahead optimization will be applied, the goal in the area surveillance problem at every discrete time step is to maximize the sum of the probabilities of the grid points that the mobile-sensors have within their sensing range. In other words, achieving maximum coverage at every discrete time step. Equation 1-11 defines how the sum of probabilities is defined.

$$G = \sum_{\bar{x}_i \in S} P(\bar{x}_i) \tag{1-11}$$

where $P(\bar{x}_i)$ is a function that returns the probability value of the corresponding grid point and S is shown in Equation 1-12.

$$S = \bigcup_{\kappa=1}^{N} D_{R_\kappa} \tag{1-12}$$

Therefore, at every discrete time step, every mobile-sensor has to choose the potential position in its mobility domain D_M that based on the other mobile-sensors maximizes the coverage, as shown in Equation 1-13.

$$G_{max} = \arg\max_{L} \sum_{\bar{x}_i \in S} P(\bar{x}_i) \tag{1-13}$$

where $L = \{d_1, ..., d_N \,|\, d_1 \in D_{M_1}, ..., d_N \in D_{M_N}\}$.

Safety constraint

Only defining the combination of positions that result to the maximum sum of probabilities cannot ensure that the mobile-sensors will avoid any collision. Hence, a safety constraint needs to be introduced. The safety constraint depends on the distance between two mobile-sensors. If the distance between any combination of potential positions of these mobile-sensors is less than a safety threshold (s_{th}), then the safety constraint will not allow the two mobile-sensors to select these potential positions. Assuming that the agents κ and ρ have the potential positions $d_\kappa \in D_{M_\kappa}$ and $d_\rho \in D_{M_\rho}$, respectively, and the distance between these two potential positions is less than the safety threshold s_t, then, a large penalty value is added to the sum of probabilities of the corresponding combination of potential positions. Adding a negative value will secure that the mobile-sensors will not select the corresponding potential positions as their next position since it will not result to the maximum sum of probabilities. Equation 1-14 indicates how the penalty value can be defined.

$$s_c(d_\kappa, d_\rho) = \begin{cases} \frac{-100}{|d_\kappa - d_\rho|} & \text{if } |d_\kappa - d_\rho| \leq s_t \\ 0 & \text{otherwise} \end{cases} \tag{1-14}$$

Long horizon heuristic

Furthermore, given the decentralized property of the area surveillance problem, and the fact that a one-step-ahead optimization is utilized, it is possible that a mobile-sensor will be trapped in an area that has been previously observed by a different mobile-sensor. In this case, the grid points that belong to this area will have a probability value of zero. Hence, every potential position in its mobility domain will result in a zero sum of probabilities and the mobile-sensor will not be able to define a waypoint that will drive it towards an unobserved area. This problem is the stalemate problem and a heuristic, known as long-horizon, to overcome it is given below.

The basic idea behind the long-horizon is to guide the mobile-sensor towards its closest area that has not been fully surveyed yet. More clearly, towards its closest area that has a sum of probabilities larger than zero. Therefore, the first step in this procedure is to split the environment into sub-areas.

Using the total number of grid points, the environment is split into sub-areas, $sub_i \in D_{sub}$, that consist of the same amount of grid points. To decide the size of the sub-areas, the sensing range of the mobile-sensors is utilized. To be more specific, the area is split into regions with the same size as the sensing range of the mobile-sensors. Moreover, every sub-area has two features. The location of the center of the sub-area (d_{sub_i}), which is the center-point of all grid points, and the sum of probabilities of the grid points that the sub-area contains (p_{sub_i}). The p_{sub_i} is defined as shown in Equation 1-15.

$$p_{sub_i} = \sum_{\bar{x}_i \in sub_i} P(\bar{x}_i) \tag{1-15}$$

Furthermore, the next step is to define the sub-area of the environment that is closest to the current position of the mobile-sensor and has a sum of probabilities larger than zero. Therefore, first, the sub-areas that have a sum of probabilities larger than zero are picked. Equation 1-16 shows how to define these sub-areas.

$$D_z := \{sub_i \in D_{sub} | \, p_{sub_i} > 0\} \tag{1-16}$$

In addition, the sub-area that is closest to the current position of the mobile-sensor is defined using Equation 1-17.

$$d_c = \underset{d_{sub_i} \in D_z}{\arg\min} |d_{sub_i} - d_\kappa| \tag{1-17}$$

Finally, the mobile-sensor can use the center of its closest sub-area to decide which one of its potential positions will drive it closest to this area. Using Equation 1-18, the mobile-sensor can compute a weight for every potential position in its mobility domain $d_\kappa \in D_{M_\kappa}$.

$$w_{d_\kappa} = \frac{1}{|d_c - d_\kappa|} \tag{1-18}$$

Equation 1-18 defines that the sub-area that is closest to the mobile-sensor will get the highest weight.

Apply long horizon heuristic

The long horizon heuristic will be applied whenever the sum of the probabilities of the grid points in the active domain D_{A_κ} of the mobile-sensor κ is zero. In this case, the mobile-sensor will consider for every potential position in its mobility domain what the corresponding weight is. Specifically, it will add the value of the weight to the corresponding sum of probabilities. So, in the end, the mobile-sensor will choose the potential position with the highest sum of probabilities, and it will drive it towards its closest area that still needs to be surveyed. The long horizon utility value can be defined using Equation 1-19.

$$w_{d_\kappa} = \begin{cases} \frac{1}{|d_c - d_\kappa|} & \text{if } \sum_{\bar{x}_i \in D_{A_\kappa}} P(\bar{x}_i) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{1-19}$$

Mobile-sensors goal

Finally, by considering the area surveillance problem, every mobile-sensor has to compute a waypoint at every discrete time step, with the main goal to achieve maximum coverage of the area. Therefore, the mobile-sensors that share a utility constraint have to communicate and coordinate their actions in order to maximize the global utility function (Equation 1-13). This will allow them to decide the combination of positions that result to the maximum coverage, based on the mobility, sensing and safety constraints. Furthermore, in the case where the sum of probabilities in the active domain of any of the mobile-sensors is zero, then the long-horizon optimization will be utilized. In this case, the mobile-sensors that share a utility constraint have to decide the combination of positions that result to the maximum coverage, based also on the long-horizon weight. Therefore, the utility value for every possible combination of position assignments can be computed using Equation 1-20.

$$U = \sum_{\bar{x}_i \in S} P(\bar{x}_i) + \sum_{\kappa=1}^{N} \sum_{\rho=\kappa+1}^{N} s_c(d_\kappa, d_\rho) + \sum_{\kappa=1}^{N} w_{d_\kappa} \tag{1-20}$$

Finally, the value assignment that result to the maximum global utility can be computed by maximizing Equation 1-21.

$$W = \arg\max_{L} U \tag{1-21}$$

where $L = \{d_1, ..., d_N \,|\, d_1 \in D_{M_1}, ..., d_N \in D_{M_N}\}$.

1-3 Thesis goal

The goal of this thesis is to derive a decentralized algorithm for solving the area surveillance problem using a group of mobile-sensors. The decentralized algorithm should be able to run on the onboard computers that the mobile-sensors carry. Due to the fact that onboard computers are not powerful computers, there are time and memory limitations. Moreover, the algorithm should be scalable with respect to the number of the mobile-sensors. That is, the algorithm should be able to define the solution to the problem independently of the number of mobile-sensors. Finally, the algorithm should be able to result the maximum coverage that the mobile-sensors can achieve at each discrete time step in order to minimize the required discrete time steps to solve the problem.
This thesis begins by introducing a coordination framework for multi-agent systems known as a Distributed Constraint Optimization Problem (DCOP). In addition, using the corresponding coordination framework, the area surveillance problem will be formulated as DCOP. Furthermore, chapter 3 will introduce several available solvers for DCOP, in order to define the most appropriate solver for solving the area surveillance problem. Moreover, chapter 4 will describe how the algorithm that has been selected from the previous chapter solves the problem and what its main limitation is. Furthermore, chapter 5 will describe a solution for overcoming this main limitation. Finally, several experiments are performed to compare the performance of the solvers and whether they are able to solve the area surveillance problem.

Chapter 2

Distributed constraint optimization problem

The multi-agent systems have been studied intensively the last few decades. Due to the fact that the agents can take actions autonomously based on their perception of their surrounding environment makes them a valuable tool for solving distributed problems. In the multi-agent systems, the agents are able to find a solution to a problem that will be impossible for a single agent. To achieve this, the agents have to cooperate with the other agents, and by incorporating their perception of their surrounding environment, they can successfully find a global solution to the problem. Making the agents able to cooperate and coordinate their actions in order to achieve a global solution is one of the most challenging parts in solving distributed problems [11]. Therefore, a coordination model that aims to coordinate the actions of the agents is needed.

In the literature, there are various approaches to define a coordination model. Some examples are reactive coordination [12], learning [13], planning [14]. However, for these approaches, it is challenging to define an efficient behavior during the interaction of the agents [11]. Another coordination model is the Distributed Constraint Optimization Problem (DCOP). The DCOP has the ability to model a large class of real world problems. Moreover, due to the ability of DCOP to define the problem by utilizing the constraints between the agents, it has emerged as one of the most important formalism for coordination [11, 14, 15]. There are several examples of multi-agent systems that used DCOP as a formalization structure, such as for sensor networks [16, 17], traffic control [18], meeting and task scheduling [19]. Moreover, in the literature, there are various available solvers that can be used for solving a DCOP [1, 20, 21]. Based on the area surveillance problem, the algorithm that should be designed needs to be able to solve the problem in a decentralized manner. Consequently, since the available DCOP solvers satisfy this requirement, the DCOP framework can be employed for the area surveillance problem.

This chapter will first introduce the definition of the DCOP. That is, the basic notations and how to represent the utility constraints between two different mobile-sensors. Furthermore, this chapter will formulate the area surveillance problem as DCOP. Finally, a brief introduction on what a dynamic DCOP is, and how to solve such a dynamic DCOP.

2-1 DCOP definition

A Constraint Optimization Problem (COP) consists of variables and constraints between these variables. In order to solve a COP, a value assignment should be defined for every variable based on the constraints, and such that the global objective is optimized. The DCOP is an extension of the COP, where the variables are distributed among agents. In the DCOP, every agent performs local computations with the main purpose of optimizing a global objective.
The DCOP consists of a set of variables and constraints, where every variable is assigned to an agent. Each variable has a discrete domain with finite values where these values represent potential states of the corresponding variable. Only the assigned agent is able to manipulate the value of its variable where it also knows the potential states of the the other variables that its own variable has constraints with. Furthermore, a constraint can be described by a utility value between a pair of variable assignments. Finally, the DCOP intends to maximize or minimize the total utility value of the constraints [11].
In the DCOP framework, only the assigned agent can adjust the values of its variables. Therefore, to optimize the sum of the resulting constraints agents have to exchange information with the agents that share a constraint, in order to coordinate their actions of assigning the optimal values to their variables.

2-1-1 DCOP model

A DCOP can be described as a tuple $< X, D, F, A, G >$ where X is a set of decision variables, D is a set of discrete domains, A is a set of agents, F is a set of constraints and G is the global utility function [22, 11]. Table 2-1 gives an overview for every set in DCOP.

Table 2-1: Description of every set in DCOP.

Set	Description
X	$X = \{\mathbf{x_1}, ...\mathbf{x_n}\}$ is a finite set of decision variables. $\mathbf{x_j}$ is the value assignment of variable j.
D	$D = \{D_1, ..., D_n\}$ is a set of finite domains for each decision variable. $D_j = [d_j^1, d_j^2, ..., d_j^q]$ where d_j^q is the q_{th} potential state of decision variable j.
A	$A = \{\alpha_1, ..., \alpha_p\}$ is a finite set of agents. α_i represents agent i.
F	$F = \{f_1, ..., f_m\}$ is a finite set of constraints.
G	Is the global utility/cost function that needs to be optimized. Usually, is the sum of the constraints $G = \sum_{j=1}^{m} f_m$

2-1-2 DCOP outcome

The outcome of the DCOP is a value assignment for every variable in the problem. These values are selected from the discrete domains of every variable, such that the sum of the constraints is maximized, Equation 2-1.

$$X^* = \arg\max_X G \tag{2-1}$$

where,

$$X^* = \{d_1^*, ..., d_n^* | d_1^* \in D_1, ..., d_n^* \in D_n\} \tag{2-2}$$

2-1-3 Representation

The DCOP can be represented as a constraint or factor graph. A constraint graph consists of nodes and edges [23]. For the DCOP, the nodes of the graph correspond to the agents and the edges to the constraints or relations between the agents. Two agents are said to be neighbors if they are connected with an edge. A simple constraint graph is depicted in Figure 2-1.

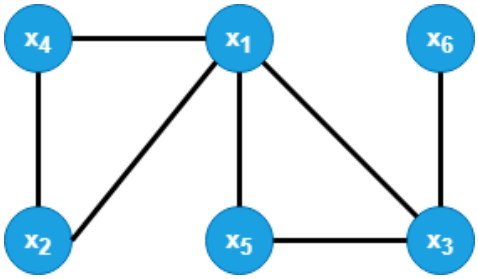


Figure 2-1: An example of a constraint graph.

Furthermore, factor graphs consist of variable nodes and function nodes [24]. In DCOP, the variable nodes are the agents and the function nodes are the constraints. A function node is connected only to the variable nodes that participate in the constraint. Two agents are said to be neighbors if they are connected to the same function node. An example of a factor graph can be seen in Figure 2-2.

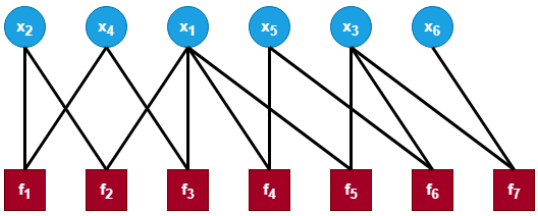


Figure 2-2: An example of a factor graph.

2-2 Area surveillance as DCOP

The DCOP framework can be utilized to represent any constraint optimization problem. Due to its simplicity and its ease of use, several solvers have been designed for this framework. Every DCOP solver is able to solve any problem that is formulated as a DCOP. That makes the DCOP framework attractive for every multi-agent systems. Additionally, a portion of DCOP solvers is capable of solving problems in a decentralized manner. That is, they have been designed such that the agents can define the optimum value assignment of their variables using their onboard computers, by exchanging messages between them. Regarding the area surveillance problem, the mobile-sensors have to perform computations on their onboard computers using information that they can get from the mobile-sensors that share a utility constraint. This implies that for solving such a problem, a decentralized approach is preferred. Furthermore, similar problems [7, 25, 26] have been formulated as DCOP with high success. Consequently, since the DCOP framework has been successfully used in the past for similar problems and it also has available decentralized solvers, is the one that will be adopted as a coordination model for the area surveillance problem.
In order to model the area surveillance problem as a DCOP, the decision variables needs to be specified. The main goal in the area surveillance is to maximize the sum of the probabilities that the mobile-sensors have within their sensing range by adjusting the position of every mobile-sensor. Therefore, the position of every mobile-sensor can be modeled as the decision variable. Based on the area surveillance problem, a decision variable $\mathbf{x}_\kappa$ will hold the $[x_\kappa, y_\kappa]$ position of the mobile-sensor κ.
Moreover, since the position of the mobile-sensor will be the decision variable, the domains will be the potential positions of the corresponding mobile-sensor. As mentioned in the previous chapter, this domain is known as the mobility domain (D_M). Consequently, a decision variable can get any value that is included in the mobility domain.
Furthermore, since the DCOP aims to maximize the sum of the utility value of the constraints, two mobile-sensors will share a constraint in the DCOP if they share a utility constraint. The reward in this case will be determined using the coverage that the two mobile-sensors can achieve based on their constraints. Equation 1-20 will be utilized for computing the utility value of a utility constraint. Therefore, using the active domains for every agent (D_A), it can be specified whether two agents share a constraint. Additionally, an agent will be responsible for only one mobile-sensor. Therefore, the agent can only set the position of only one mobile-sensor. For the remainder of this thesis, the mobile-sensor κ will be symbolized as an agent α_κ, since there is one-to-one relation.
Finally, the main goal of DCOP is to maximize the total coverage. This can be achieved by maximizing the sum of the constraints. Hence, the outcome for every mobile-sensor will be its next position such that they achieve maximum coverage of the area. Table 2-2 summarizes how area surveillance is formulated as DCOP.

Constraint graph for area surveillance

The most common way to represent the constraints between agents in DCOP is by using a constraint graph. For the area surveillance problem, nodes represent the mobile sensors and edges the utility constraints. Therefore, two mobile-sensors will be connected with an edge in the constraint graph if their active domains overlap. In addition, two mobile-sensors are

Table 2-2: Formulate the area surveillance as a DCOP.

Set	Description
$X = \{\mathbf{x_1}, ..., \mathbf{x_N}\}$	$\mathbf{x}_\kappa = [x_\kappa, y_\kappa]$ of mobile-sensor κ.
$D = \{D_1, ...D_N\}$	D_κ is the mobility domain D_{M_κ} of mobile-sensor κ.
$A = \{\alpha_1, ...\alpha_N\}$	An agent α_i is responsible for only one decision variable.
$F = \{f_1, ...f_m\}$	f_m is a utility constraint between two agents and it defines the coverage that both can achieve. A utility constraint exists if the active domains of two mobile-sensors contain the same grid points.
G	Is the sum of utility constraints.

neighbors if they share a utility constraint.

Multiple constraint graphs

An example of a constraint graph for the area surveillance problem can be seen in Figure 2-3. This figure shows on the left side the agents along with their active domains and on the right side the constraint graphs that corresponds to that formation. As it can be seen from this figure, the active domains of agents α_4 and α_5 do not overlap with the active domains of the other agents. This means that the agents α_4 and α_5 will not share any constraint with the rest. Therefore, two distinct constraint graphs occur. When multiple constraints graph exist at the same discrete time step, every graph represents a separate area surveillance problem since all of them are independent. Therefore, the area-surveillance problem can be split into sub-problems. Additionally, every sub-problem has to define the global utility that its mobile-sensors can achieve. Concluding, the total coverage of the whole system will be the summation of every sub-problem. For example, the total coverage of the area surveillance problem in Figure 2-3 will be the summation of the two sub-problems.

2-3 Dynamic DCOP

DCOP has been designed as a coordination model for static problems [27]. However, as most of the real world applications problems do change over time, with respect to the constraints and the values of the variable, the area surveillance problem does as well. Not only because of the dynamic behavior of the environment but also because of the ability that the mobile-sensors have to navigate around the environment. Therefore, at every discrete time step, the neighbors of a corresponding mobile-sensor might differ. This will result in different constraint graphs at every discrete time step, and therefore a different DCOP will have to be solved. This is known as the dynamic DCOP.
Yeoh et al. [27] proposed an algorithm that aims to use as much as possible information from the previous DCOP, in order to save some time on re-constructing the new DCOP. Another approach to cope with the dynamic DCOP, is to consider the dynamic DCOP as a sequence

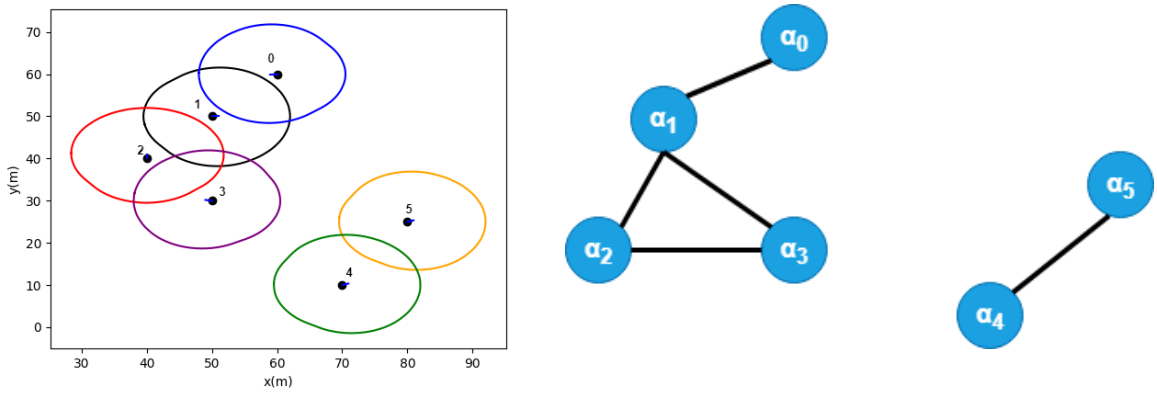


Figure 2-3: Left: Represents the current formation of the mobile-sensors. Black dots represent their position. The contour lines represent their active domains. Right: A constraint graph for the given formation where nodes are the agents and edges are the utility constraints.

of DCOPs where each DCOP is different from the DCOP proceeding it [27]. More precisely, to define at every discrete time step, the new domains and the new constraints between the agents, and to solve the DCOP based on them.
In the area surveillance problem, it is not reasonable to re-use information for reconstructing the DCOP. This is mainly due to that fact that at every discrete time step the constraints between the mobile-sensors are not the same, the domains of the mobile-sensors are different, where also the probability of every grid point of the environment changes. Consequently, a sequence of DCOPs will be used for solving the dynamic DCOP.
Using a sequence of DCOPs implies that at every discrete time step, the mobile-sensors have first to define their neighbors and then formulate the DCOP. Therefore, a different DCOP has to be solved at every iteration.

2-4 Conclusion

This chapter first introduced the DCOP framework. DCOP provides a useful framework for coordinating multi-agent systems in centralized and decentralized manner. Furthermore, its available solvers are suitable for any problem that is formulated as DCOP. Therefore, the DCOP has been selected as the coordination model for the area surveillance. Concerning the area surveillance problem, the positions of the mobile-sensors have been modeled as the decision variables. Moreover, mobile-sensors can get in the next discrete time step any position that is included in their mobility domain. Therefore, the mobility domain represents the domain for every decision variable. Finally, the DCOP aims to maximize the global utility function, in the case of the area surveillance problem the sum of the utility constraints, by choosing for every decision variable the value in their domain that it achieves this.
Due to the ability of DCOP to successfully represent multi-agent systems, several researchers designed solvers for solving these type of problems. However, choosing the correct solver for solving the corresponding problem is challenging. Consequently, the next chapter will briefly explain some of the available DCOP solvers in order to decide which one suits best for the area surveillance problem.

Chapter 3

DCOP solvers

The Distributed Constraint Optimization Problem (DCOP) is a framework that is suitable for multi-agent systems. All the variables and the constraints of the multi-agent system are defined within this framework. Computing a solution for a DCOP means to find a value assignment for every agent in the DCOP such that they achieve to maximize the global utility function. Therefore, a DCOP solver aims to calculate the optimum value assignment for every agent such that a global function is optimized. A DCOP solver can be classified into centralized and decentralized. Centralized solvers have a single agent, the coordinator, to decide the final value assignment for every agent, using information from the other agents. On the other hand, in the decentralized solvers, the agents using their onboard computers determine the optimum value assignment of their assigned variables by exchanging messages with the other agents. The centralized approach introduces a single point of failure, where if the coordinator fails, the agents will not be able to define their optimum value assignment. Moreover, solving a DCOP in a centralized manner is NP-hard problem [28]. Hence, the coordinator requires a lot of computation power to define the optimal solution. On the contrary, in the decentralized approach there is no single point of failure, where also the required computational power is split to every agent. Based on the area surveillance problem, and due to the restricted available resources of the mobile-sensors, a centralized approach will be infeasible. Moreover, Lian et al. [29] proved that a decentralized solver can have the same or even better convergence rate with a centralized one. For these reasons, only decentralized solvers will be studied for the remained of this thesis.

In the literature, various decentralized DCOP solvers can be found which can be categorized into two main categories: the complete and incomplete solvers. Complete solvers guarantee to find the optimal solution. However, they require exponential communication and/or computation based on the number of agents [1, 20]. Therefore, complete algorithms are not well suited for multi-agent systems. On the other hand, incomplete solvers have faster convergence than the complete ones but they achieve this by finding a sub-optimal solution [11, 30].

Finally, the DCOP solvers differ also on their synchronization scheme. There exist two type of synchronization schemes, synchronous and asynchronous. In the synchronous solvers, an agent has to wait for a message from the agents that share a constraint before changing the values of its assigned variables [20]. On the contrary, in the asynchronous solvers an agent

does not have to wait to change the values of its assigned variables [1].
Figure 3-1 shows solver categories along with some example of solvers. This chapter will first give an overview of the DCOP solvers categories and their main differences. This overview will lead to understand their main limitations. In addition, this chapter will introduce the solver requirements for the area surveillance problem. Using the solver's requirements, the solver that best fits the requirements will be employed for the area surveillance problem.

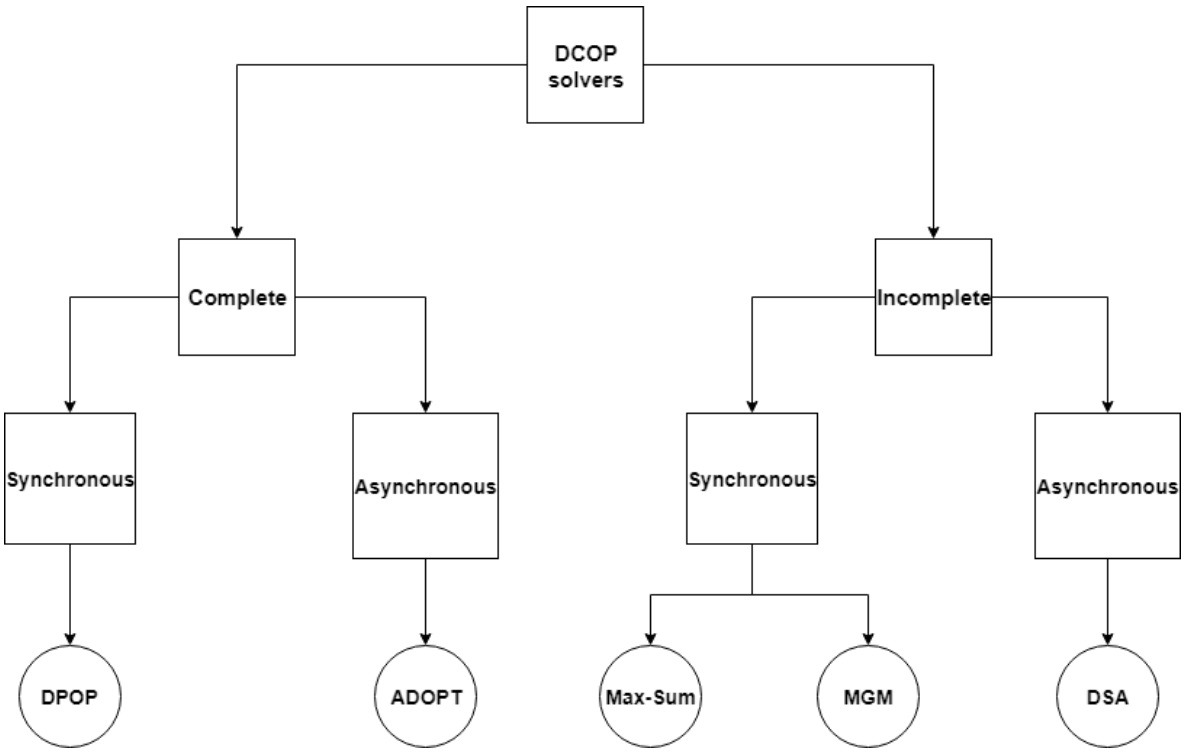


Figure 3-1: DCOP solvers categories. The squares show the solvers categories and the circles indicate some of the available solvers.

3-1 Complete solvers

Complete solvers guarantee that their solution will be the global optimum, that is, the best feasible solution. The complete solvers can be divided into two classes [11]: the inference based, and the search based.
Search-based are efficient in that the search can be terminated without searching the full space. On the contrary, inference-based can reduce the number of messages by sending aggregated utility. Two of the most well-known complete solvers are the Distributed pseudo-tree-optimization procedure (DPOP), an inferenece based algorithm, and the Asynchronous

Distributed OPTimization (ADOPT), a search-based algorithm. Both solvers operate over a pseudo-tree for defining the optimum solution. For this reason, the definition of the pseudo-tree will be given next.

Pseudo-tree

A pseudo-tree is a re-arrangement of the constraint graph where it has the same nodes and edges as the constraint graph. The definition of the constraint graph for the area surveillance problem can be found in Section 2-2. Moreover, a pseudo-tree has the property that adjacent nodes in the constraint graph fall in the same branch in the pseudo-tree structure [30, 20, 11]. One of the most well-known algorithms for deriving the pseudo-tree is the graph traversal method Depth-First Search (DFS) [31]. DFS requires every agent to know its neighbors, that is, the agents that they share a constraint. Moreover, the DFS algorithm requires as a parameter, an agent that will first start the DFS procedure, known as the root of the pseudo-tree. There are several heuristics to decide the root agent. The most well-known heuristic is to choose the agent that has the most neighbors. The outcome of the DFS is a label for every neighbor of the agent. More specific, during the DFS procedure, every agent is able to classify its neighbors in one of the following categories: parent, pseudo-parent, children, pseudo-children.

DFS algorithm

Algorithm 1 shows the DFS procedure. DFS algorithm uses a "token procedure" to classify its neighbors in one of the abovementioned categories. Initially, the root agent starts the token procedure by adding its name in the token. Then, it sends the token to one of its unvisited neighbors, and it waits to receive it back before sending it to its other unvisited neighbors. The receiver agent α_i first marks the sender as its parent. In addition, if any of its neighbors are included in the token, it marks them as pseudo-parents. After this, the agent α_i adds itself in the token, and it sends the token in turn to each one of its unvisited neighbors α_j. These agents become the children of the agent α_i. Whenever an agent receives the token from one of its neighbors, it marks the sender as visited. Finally, the token can return to the agent α_i either from the agent α_j, the neighbor that previously sent the token, or from a different neighbor, the agent α_k. In the case that the token is returned from the agent α_k, then it means there is a cycle in the tree, and the α_i marks the agent α_k as its pseudo-child.

Algorithm 1: DFS algorithm for constructing the pseudo-tree [30]

Data: neighbors of every agent.
Result: each agent labels all its neighbors as either P, PP, C, PC

Procedure Token Passing(performed by each agent α_i)
if α_i is root **then**
  $P(\alpha_i) = null$
  create empty token DFS
else
  $DFS =$ **HandleIncomingTokens()**
end if
let $DFS_{\alpha_i} = DFS \cup \{\alpha_i\}$
for all $\alpha_l \in N_{\alpha_i}$ **do**
  if α_l not visited yet **then**
    add α_l to $C(\alpha_i)$
    send DFS_i to α_l
    wait for DFS_i to return from α_l
  end if
end for

Procedure HandleIncomingTokens()
wait for any incoming DFS_l message
let the agent α_l be the sender
mark the agent α_l as visited
if this is the first DFS message (agent α_l is my parent) **then**
  $P(\alpha_i) = \alpha_l$
  $PP(\alpha_i) = \{\alpha_k \neq P(\alpha_i) | \alpha_k \in N_{\alpha_i} \cap DFS_l\}$
else
  if $\alpha_l \in C(\alpha_i)$(this is a DFS message returning form a child) **then**
    continue with other neighbors
  else
    (the message is DFS message coming from a psuedo-child)
    add the agent α_l to $PC(\alpha_i)$
  end if
end if

Pseudo-tree example

Figure 3-2 shows an example of the construction of the pseudo-tree. In the pseudo-tree arrangement, the solid lines represent a parent-child relationship and the dashed lines a pseudo-parent - pseudo-child relationship. Below, the explanation of every category is given. The given examples will be based on Figure 3-2.

- $P(\alpha_i)$ is the parent of the agent α_i. This is the ancestor of the agent α_i which is connected to α_i through a tree edge. For example, the agent α_2 is the parent of the

agent α_8 ($P(\alpha_8) = \alpha_2$).

- $C(\alpha_i)$ are the children of the agent α_i. These are the descendants of agent α_i which are connected to α_i through tree edges. For example, the agents α_7 and α_8 are children of the agent α_2 ($C(\alpha_2) = \{\alpha_7, \alpha_8\}$).

- PP(α_i) are the pseudo-parents of the agent α_i. These are the ancestors of agent α_i that are connected to α_i through back edges. For example, the agent α_1 is the pseudo-parent of the agent α_5 ($PP(\alpha_5) = \alpha_1$).

- PC(α_i) are the pseudo-children of the agent α_i. These are the descendants of agent α_i that are connected to α_i through back edges. For example, the agent α_9 is a pseudo-child of the agent α_2 ($PC(\alpha_2) = \alpha_9$).

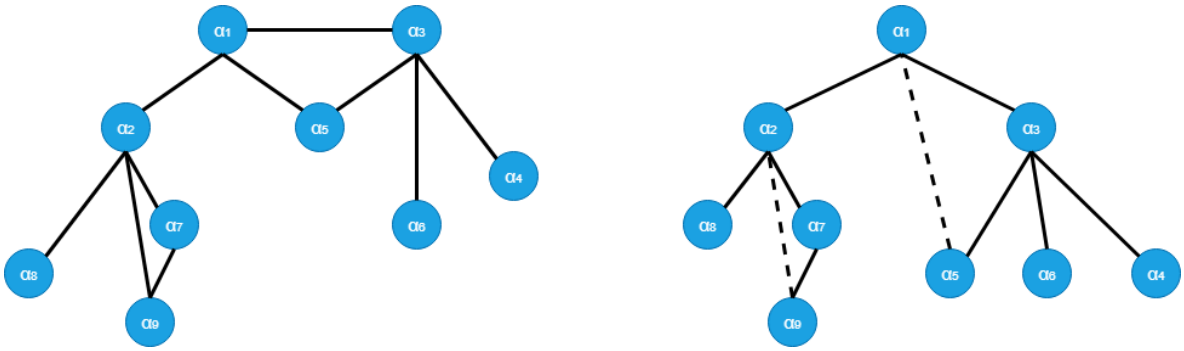


Figure 3-2: An example of deriving the pseudo-tree from the given constraint graph. The constraint graph is shown on the left and the pseudo-tree can be seen on the right.

Additionally, another three important definitions for the pseudo-tree are the definitions of the leaves, the separator and the induced width of the pseudo-tree. These three definitions will be utilized during the explanation of the DPOP and the ADOPT solvers. Specifically, the separator and the induced width will be utilized to explain the complexity of the solvers.
First, the leaves are the agents that do not have any children or pseudo-children. In Figure 3-2 the agents $\alpha_4, \alpha_5, \alpha_6, \alpha_8, \alpha_9$ are the leaves of the pseudo-tree. Furthermore, the separator is a set of agents that contains the parents and pseudo-parents of agent α_i, and the parents and pseudo-parents of its children. To define the separator of an agent α_i, Equation 3-1 is utilized.

$$s_i = P(\alpha_i) \cup PP(\alpha_i) \cup \Big(\bigcup_{j \in C(\alpha_i)} s_j \Big) - \alpha_i \tag{3-1}$$

For example, in Figure 3-2, the separator of the agent α_3 is $s_3 = \{\alpha_1\}$ and the separator of the agent α_9 is $s_9 = \{\alpha_2, \alpha_7\}$.
Finally, the induced width of the pseudo-tree is equal to the size of the largest separator of any agent in the pseudo-tree [30]. For example, the induced width of the pseudo-tree in Figure 3-2 is equal to 2.

3-1-1 Inference-based solvers

Inference-based solvers aim to compute the optimal solution by propagating the aggregated costs of constraints. To get an insight into how inference-based solvers work, a brief explanation of the inference based solver DPOP will be given.
The DPOP was first introduced by Petcu and Faltings [20]. Based on Petcu and Faltings [20, 30], the DPOP requires a linear number of messages to solve the problem by using dynamic programming techniques. The DPOP consists of three phases. The first phase is the construction of the pseudo-tree, then, is the utility propagation and finally is the value propagation. A brief explanation of every phase will take place below.

Pseudo-tree construction

The DPOP operates over a pseudo-tree, hence, it starts by creating the pseudo-tree from the constraint graph. This can be achieved by using the DFS algorithm (Algorithm 1). The pseudo-tree is used as a communication structure for the next two phases. The agents only exchange messages with agents that are connected through the tree edges. One of the main advantages of using the pseudo-tree as a communication structure is that agents that lie in different branches are able to operate in parallel. This happens due to the fact that these agents do not share any utility constraint and therefore, their solution is independent from each other. For example, agents $\alpha_4, \alpha_5, \alpha_6, \alpha_8, \alpha_9$, from Figure 3-2, can start the utility propagation phase at the same time.

Utility propagation

Utility propagation phase starts from the leaves, and it propagates up the pseudo-tree through the tree edges. In this phase, the leave agents start by creating the utility messages that they have to send to their parent. The utility message provides to the parent, the maximum utility that the child agent and its sub-tree can achieve based on the rest of the problem. More specifically, a utility message contains the utility value for each set of value assignments of the agents that are including in the separator of the sending agent. Additionally, the sending agent has to define for every possible value assignment of the agents in its separator, the optimal value assignment for itself. This operation is known as projection operation. Then, after the sending agent has projected out itself from the utility message, the sending agent is ready to send the utility message to its parent. Furthermore, as soon as an agent has received all of the utility messages from its children, it can start the utility propagation phase. Finally, the utility propagation phase ends as soon as the root agent receives every utility message from its children.
Figure 3-3 shows an example of the utility propagation phase. Agents α_1 and α_2 are the leaves in this pseudo-tree. Therefore, since they do not have any children so they have to wait for any utility messages, they start by first creating the utility matrix, and then by following the projection operation, they create the utility messages U_1^0 and U_2^0. The utility message U_1^0 represents the utility message that is sent by agent α_1 to the agent α_0. The optimum utility of agents α_1 and α_2 for every value assignment of their parent is depicted in Figure 3-3 with the shaded colors. Then, both agents send their utility messages U_1^0 and U_2^0 to their parent

agent α_0 and the utility propagation phase is done because the root agent has received every utility message from its children.

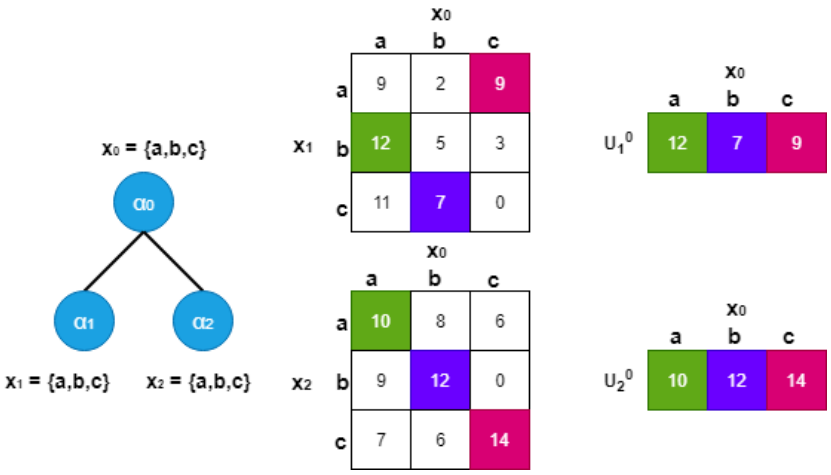


Figure 3-3: An example of the utility propagation phase. In this example, both agents α_1 and α_2 starts at the same time to create the utility message for their parents. The two matrices represent the utility messages before agents α_1 and α_2 follow the projection operation. The shaded colors in the matrices represent the optimum utility that the agents can achieve based on the utility constraints with their parents. Finally, the two tables on the right represent the utility messages that agents α_1 and α_2 send to their parent agent α_0.

Value propagation

The value propagation phase starts after the root of the pseudo-tree has received every utility message from its children. In the value propagation phase, the root starts by sending a value message to its children. A value message contains the optimal value that the corresponding agent can achieve with respect to the utility constraints between its children. Since the root agent α_r has accumulated the combined utility values, it is able to define its own optimal value assignment of its variables ($\mathbf{x_r^*}$) that results to the maximum utility. Then, the root agent α_r append the optimal value assignment to the value message $V_r^{C(\alpha_r)} = \{\mathbf{x_r^*}\}$, and it sends this message to all of its children. The $V_r^{C(\alpha_r)}$ symbolizes the value message that is sent by the root agent α_r to its children $C(\alpha_r)$.
In addition, as soon as an agent receives a value-message from its parent, it is able to define its own optimum value $\mathbf{x}_\kappa^*$. Then, the agent appends its own optimum value $\mathbf{x}_\kappa^*$ to the value message $V_\kappa^{C(\alpha_\kappa)} = \{\mathbf{x_r^*}, ..., \mathbf{x}_\kappa^*\}$, and it sends the value-message to its children. The value propagation ends when value messages reach the leaves of the pseudo-tree.

3-1-2 Search-based solvers

Search-based solvers aim to define the optimum solution, by using techniques such as best-first, backtracking, and branch and bound [11]. The ADOPT is the first distributed, search-based and complete solver. After the ADOPT has been introduced, several researchers made use

of its communication framework and message passing [32]. Therefore, in this subsection, a brief explanation of the ADOPT solver will take place in order to understand how the search based solvers operate for defining the optimal solution.
The ADOPT is an asynchronous algorithm that uses bound propagation technique for computing the optimal solution [1]. The ADOPT operates over a pseudo-tree, where its agents exchange three type of messages: value messages, cost messages, and threshold messages. Figure 3-4 shows an example of a communication graph where additional information of the messages can be found below.

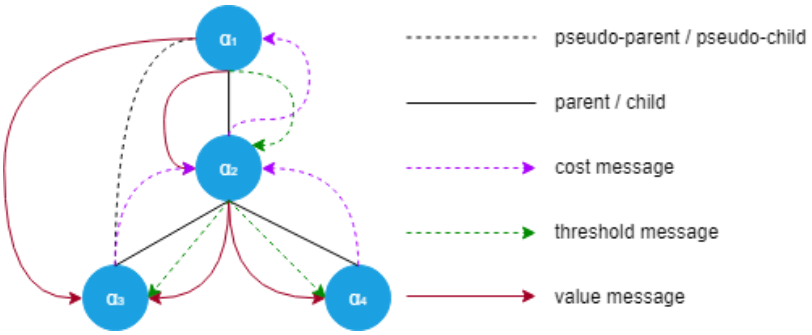


Figure 3-4: The communication structure of ADOPT solver [1].

Messages

A value message contains the agent's current value. An agent sends a value message to its children and pseudo-children after it has changed the value assignment of its variables. Furthermore, a cost message contains the following: the context, the lower bound and the upper bound. The context holds the value assignment for every agent that was used for computing the current lower bound and upper bound. The lower bound is the minimum local utility of the current agent plus all the lower bounds of its children, where local utility is the sum of utilities from constraints between the current agent and its parent/pseudo-parents. Upper bound is the maximum local utility plus all the upper bounds from its children. An agent sends a cost message only to its parent and after it has received a value message from its parent or pseudo-parent. Finally, in order to reduce redundant search, an agent sends to its children a threshold value through the threshold message. A threshold message reduces the redundant search since it does not allow the receiver to consider solutions that result to lower bound lower than the threshold. The threshold message is utilized to reconstruct solutions that were rejected before as sub-optimal.

ADOPT algorithm

The first step in the ADOPT is to create the pseudo-tree from the constraint graph. The pseudo-tree can be derived using the DFS algorithm (Algorithm 1). Then ADOPT performs a distributed backtrack search using the best-first search. During the best-first search, each agent chooses the value assignment of its variables that returns the optimal local utility. Initially, each agent randomly selects its initial value assignment. In addition, the agent

sends its optimal value assignment of its variables to its children and pseudo-children and it waits for a cost message from its children. Furthermore, when an agent receives a cost message from its children, it adds the cost values to its lower and upper bounds. Moreover, the agent checks whether there is another value in its domain that results in smaller lower bound. If this is the case, the agent changes its value and the process repeats.
However, by following this asynchronous procedure, the agents might abandon solutions before even proof that they are sub-optimal. Hence, agents may need to reconstruct previously explored solutions. To do so, ADOPT stores the lower bounds as backtrack thresholds. Then, a parent agent sends this backtrack threshold to its children, so its children do not consider any solution that will result in a lower bound lower than the backtrack threshold. Since this backtrack threshold is a lower bound that has been calculated before, it can ensure that the cost of the optimal solution will be higher or equal to the backtrack threshold. Hence, using the backtrack threshold, the ADOPT guarantees that the optimal solution will never be missed.
Finally, ADOPT uses the bound interval, the difference between the lower bound and the upper bound, to determine whether the algorithm has to terminate. If the lower bound is equal to the upper bound then the optimal solution has been determined. Furthermore, one of the main advantages of using the bound interval is that a sub-optimal solution with a certain error-bound can be found. That is, by adjusting the desired difference between the lower and the upper bound, pre-defined threshold, the algorithm terminates as soon as the difference between the lower and the upper bound is less or equal to the pre-defined threshold. If the pre-defined threshold is equal to zero, then ADOP will define the optimum solution. If the pre-defined threshold is larger than zero, the ADOP will compute a sub-optimum solution with an error equal to the pre-defined threshold.

3-1-3 Comparison

Both inference-based solvers and search-based solvers guarantee to find the optimal solution of the problem in a distributed manner. The ADOPT is an asynchronous solver, where the idle time of the agents is less than the DPOP due to the fact that the agents can change their value depending on their local view of the problem. However, because of the asynchronous manner of the ADOPT, the agents have to exchange a large number of small messages to define the optimal solution. More specifically, in the worst case scenario, the ADOPT requires an exponential number of messages with respect to the height of the pseudo-tree to achieve the optimal solution. Considering the area surveillance problem, this will create communication overhead since the mobile sensors will have to spend time on communicating with their neighbors for defining the optimum solution.
On the contrary, the DPOP solver is using the utility propagation technique to define the optimal solution. By doing this, it reduces the number of messages that the agents have to exchange. More precisely, the number of messages that DPOP requires to achieve the optimal solution is linear in the number of agents. This is because messages are propagating first from the leaves to the root and then from the root to the leaves. However, the main disadvantage of the DPOP lies in the size of the messages. The largest message of DPOP will be space exponential in the induced width of the pseudo-tree. Therefore, the higher the induced width of the pseudo-tree, the larger the message, and consequently the larger the amount of memory that DPOP requires. Regarding the area surveillance problem, and due to the fact the mobile

sensors are equipped with an onboard computer that has a limited amount of memory, the algorithm may not be able to compute the optimal solution.
By comparing these two type of solvers, both can return the optimal solution in a decentralized manner. However, both of them are not scalable with respect to the number of agents since they require either exponential size of messages or exponential number of messages for defining the solution. Hence, based on the limited computation resources that the mobile-sensors have and the exponential requirements of both solver, both solvers may fail to define the solution of the problem if the number of mobile-sensors is high.

3-2 Incomplete solvers

In contrast to complete solvers, incomplete solvers do not guarantee that the outcome will be the global optimum. However, to find the solution of the DCOP, the incomplete solvers require less computation and communication cost compared to the complete solvers. Incomplete solvers can be further divided into three categories: the local search, the inference-based, and the sampling-based [11].
The local search solvers compute the solution using only the local information of the agents, where the incomplete inference based solvers, as the complete inference based do, compute the accumulated utility of constraints. The sampling-based solvers sample the search space to approximate a function as a product of statistical inference [33]. This section, will first introduce the max-sum, an inference-based solver. In addition it will briefly describe two of the most well known local solvers. This will lead to an insight of how incomplete inference-based and local search solvers solve the DCOP.

3-2-1 Incomplete inference-based solver

The Max-Sum solver is an inference-based solver where it uses a cyclic bipartite factor graph [26]. Bipartite factor graph consists of variable and function nodes where variables correspond to the variables and functions to the utility constraints. The edges indicate dependencies between the function nodes and the variable nodes. By representing the utility constraints as function nodes, the global function is split into local functions, where the local functions depend on a subset of variables. Figure 3-5 shows an instance of a factor graph, where x_1, x_2, x_3 are the variables and F_1, F_2, F_3 are the utility constraints.

Max-sum operates over the factor graph, where variable nodes exchange messages with the function nodes. More precisely, the variable nodes send to the function nodes that are connected through an edge, their value assignment. The function nodes compute a utility value based on the constraints, and they send this utility value back to the corresponding variable node. In addition, the variable node computes the total utility value of this value assignment by summing up all the utility values that it has received from the function nodes. This procedure is repeated for every possible value assignment in the finite discrete domain of the variable node. After the variable node has computed the total utility for every value assignment of its variable, it chooses as the value assignment the value that has the maximum sum of utilities.
The max-sum algorithm can guarantee the optimal solution when is applied to a factor graph without cycles [2]. However, when the factor graph contains cycles, it returns a sub-optimal

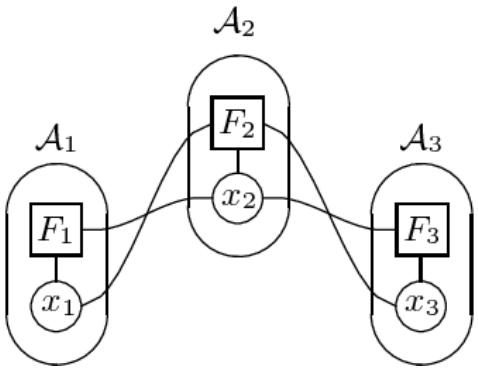


Figure 3-5: An example of a cyclic factor graph from [2].

solution. Therefore, since the area-surveillance problem may introduce cycles in the factor graph, the max-sum will not be able to define the global optimum.

3-2-2 Local search solvers

Maximum Gain Message (MGM) [21] and Distributed Stochastic Algorithm (DSA) [34] are two local search solvers which operate over the constraint graph. In both solvers, the agents compute the highest utility that their local variables can achieve based on the value assignment of their neighbors. A brief explanation of the local search solvers is given below. More specifically, every agent follows the following procedure to define its value assignment with the highest utility.

1. (a) Send value assignment to all neighbors.
   (b) Receive value assignment from all neighbors.
2. (a) Computes its highest utility value that its local variables can achieve based on assignments of neighbors.
   (b) Send the utility value to the neighbors.
3. (a) Receive the utility value from the neighbors.
   (b) Check and update the value assignment.

In the MGM solver, only the agent that has the highest utility value compared to its neighbors change its value assignment. In the DSA solver, the agents do not send their highest utility to their neighbors. The agents stochastically decide to change their values. To be more specific, if the highest utility of the agent has a positive impact on the problem, for example a utility value larger than 0, then the agent changes the value assignment of its variables with probability p. This approach, reduces the number of required messages to define the solution to the DCOP.

3-2-3 Comparison

The Max-Sum, the MGM, and the DSA are three of the most popular incomplete solvers. They compute the solution locally, by exchanging messages with the agents that share constraint using either the constraint graph or the factor graph. To define the solution, every agent determines the value assignment of its variables, that based on the value assignments of the agents that shares constraints results to the maximum utility value. However, their main limitation is that they cannot guarantee the optimal solution.
The Max-sum operates over a factor graph, where it can return the global optimum if the factor graph does not contain any cycles. Both the MGM and the DSA require a linear number of messages to define the solution. However, since they perform local computations in order to define the solution, they could get stuck in local minimum and they may result into poor solutions. The main difference between the MGM and the DSA is that in the DSA an agent can change its value assignment if it has a utility value higher than 0 with a probability p where in MGM an agent can change its value if it has the biggest utility compared to its neighbors. Therefore, the performance of the DSA depends on the probability value [34].
The Max-sum, the MGM and the DSA can successfully solve the area surveillance problem in a decentralized manner. Moreover, all the three of them are scalable with respect to the number of agents, since they do not have any exponential requirements.

3-3 Solver requirements

In order to decide which of these type of solvers best fits for the area surveillance problem, the solvers requirement have to be introduced. Based on the definition of the area surveillance problem, and given the fact that the onboard computers of the mobile-sensors have limited computation resources, a decentralized approach is preferred. Therefore, the solver should be able to find a solution in a decentralized manner by using only communication with the agents that share utility constraints, known as neighbors. Additionally, since the mobile-sensors have to survey the given area in an optimal manner, the algorithm should be able to compute the optimum solution at every discrete time step. On the other hand, when there is a time or memory limitation, the algorithm should be able to provide a sub-optimum solution that is within an acceptance range from the optimum solution. More precisely, the sub-optimum solution should be within 5% of the optimum solution. This will lead to keep the required number of discrete time steps for solving the area surveillance as low as possible. Finally, the DCOP solver should be scalable with respect to the number of agents. That is, the DCOP solver should be able to define the optimum solution for every agent even after increasing the number of agents that participate in the area surveillance problem.

Solvers comparison

The overview of the type of the solvers that this chapter introduced can be seen in Table 3-1. Based on the solver's requirements, the type of solver that satisfies all the two of them will be picked for solving the area surveillance problem. The comparison will be based on the solvers that have been introduced in this chapter.

First, a decision between the two type of complete solvers should be made. The complete solvers guarantee the optimal solution in a decentralized manner. Hence both of them satisfy the solver's requirements regarding the decentralized manner and the optimum solution. However, to define the optimal solution, both inference and search based solvers require exponential computational effort, in terms of the number of messages, or the memory required for each agent to perform its actions [11]. This implies that the solver may require a lot of computational time to derive the optimal solution, especially for large-scale applications.
Complete inference-based solvers have as its main limitation that they require a large amount of memory to store the messages if the induced-width of the pseudo-tree is high. In that case, the agents may run out of memory and be unable to perform the required computations for defining the solution. Hence, the solver will fail to define the global optimum. In contrast, the complete search-based solvers, in the worst case scenario, they require an exponential number of messages for defining the global optimum. Therefore, they can create communication overhead. That is, the agents will exchange exponential number of small messages before defining the optimum solution.
The main difference between these two types of solvers lies in the number and the size of messages that the agents have to exchange. Therefore, to decide which type of solvers, inference or search based, best fits for the area surveillance problem, a further comparison regarding the number of messages is needed. For real-world applications, it is preferable to send fewer messages with a larger size than several messages with small size [30]. By sending fewer messages with a larger size, the interaction between the agents is bounded and the communication overhead is reduced as well. It is important to reduce the communication overhead because the difference in the communication overhead can go up to order of magnitudes speedups [20]. Finally, sending fewer and larger messages can also decrease the amount of information that the agents may exchange compared to sending an exponential number of messages with small size [30]. This is mainly happen due to the fact that in the asynchronous solvers, the agents have to attach to the messages the full context of the value assignments. Exchanging less information between the agents might be vital in such applications. Despite the several extensions of the ADOPT, the number of messages is still higher than any inference-based solver. Moreover, based on the available extensions of the DPOP solver, there are several heuristics that can be employed to overcome the high memory requirements of the DPOP [35, 36, 37]. Consequently, the complete inference-based solver DPOP is preferred than complete search-based ADOPT to avoid exchanging an exponential number of small messages.
The final choice should be made between the complete inference-based, and the incomplete solvers. Incomplete solvers have the fast convergence as their main advantage. However, they cannot guarantee the global optimum, neither that the error of the solution compared to the optimum solution will be low. Consequently, the incomplete solvers do not satisfy the requirement for the optimum solution, of a solution with error less than 5%. Oppositely, the complete inference-based solvers, do not satisfy the scalability requirement since in the case of a large scale application, the agents may run out of memory and therefore the solver will be unable to define the solution. To finally decide whether a complete inference-based or an incomplete type of solver will be employed for solving the area surveillance problem, the scaling of the problem needs to be further analyzed.
As illustrated in Chapter 2, depending on the utility constraints, the problem can be divided into several sub-problems, where every sub-problem will have to solve its own DCOP. Consequently, a large scale problem can be split into several small scale problems. This infers that a complete inference-based approach can be employed for solving every DCOP separately.

Nevertheless, in the case where a large scale problem cannot be split into sub-problems, the large scale problem may result in a pseudo-tree with high induced width where the complete inference-based solver will fail to compute a solution. As mentioned above, there are several heuristics that can be applied to overcome the limitation of the exponential memory requirement that complete inference-based has [35, 36, 37]. Some of these heuristics can overcome this limitation and still guarantee the optimal solution, where others will result in computing a sub-optimum solution. Hence, by applying a heuristic along with the complete inference-based solver can make this type of solvers scalable with respect to the number of agents. However, based on the heuristic that is employed, the complete inference-based solver might violate the requirement for optimum solution.
Oppositely, the incomplete solvers do not satisfy the requirement for the optimum solution even in a small scale problems. This leads to the conclusion that the complete inference-based solver along with a heuristic for reducing the message size best fits for the solver's requirements. Consequently, the complete-inference based solver along with a heuristic for reducing the size of the message will be adopted for solving the area surveillance problem. Since the DPOP is a complete inference based solver and it has been extended in various ways, it will be employed to solve the area surveillance problem. In addition, the heuristic that will be utilized along with the DPOP will need to be designed such that it can ensure that the solver will not fail whenever the induced width of the pseudo-trees is high so that the solver will be able to define a solution to the problem.

3-4 Conclusion

In this chapter, different type of DCOP solvers have been introduced. Based on the solvers requirement and the characteristics of these type of solvers, the complete inference based approach is the one that best fits the requirements. One of the most well-known complete inference based solvers is the DPOP. However, to make the DPOP applicable to the area surveillance problem, a heuristic for reducing the size of the utility messages needs to be employed along with the DPOP. Hence, the DPOP solver along with a heuristic will be employed for solving the area surveillance. In the next chapter, the DPOP solver will be explained in more details. More specifically, the next chapter will first describe how DPOP solves the area surveillance problem and how does its main limitation affect the performance of the solver.

Table 3-1: Overview of the DCOP solvers presented in this chapter.

Algorithm	Details	Limitation
Complete inference-based (DPOP)	Synchronous manner. Return the global-optimum. Operates over the pseudo-tree Utility propagation. Minimize the number of required messages.	Requires a large amount of memory to store the messages. It could fail to return a solution due to memory requirements.
Complete search-based (ADOPT)	Asynchronous manner. Return the global-optimum. Operates over the pseudo-tree. Best-first search. Minimize the idle time of the agents.	It can result in communication overhead. It could fail to return a solution due to time limitations.
Incomplete inference-based (Max-Sum)	Return a sub-optimal. Operates over a factor graph. Maximizes the sum of utilities received from the function nodes.	Not quality on the solution. Poor results when the factor graph contains cycles.
Incomplete local search (MGM/DSA)	Return sub-optimal. Operates over a constraint graph. Maximizes the local utility. Agents change value based on its local utility. They have fast convergence.	Not quality on the solution. Agents can be trapped into local minimal.

Chapter 4

DPOP for area surveillance

Following the previous chapter, the Distributed Pseudo-tree Optimization Procedure (DPOP) [20] will be employed to solve the area surveillance problem. The DPOP is one of the most well known solvers for solving decentralized problems in a synchronous manner. It is a complete solver, where its main advantage is that it requires a linear number of messages for determining the optimal solution.
Concerning the area surveillance problem, the goal is to successfully survey a given area. To achieve this, a one-step-ahead optimization will be adopted. In the one-step-ahead optimization, the mobile-sensors have to adjust their position at every discrete time step such that they maximize the sum of the probabilities of the grid points with respect to their sensing range. This goal is also known as maximizing the current coverage. Solving the area surveillance problem using a one-step-ahead optimization is a constrained optimization, where the maximum coverage should be determined based on the mobility, sensing, and safety constraints. Additionally, the area surveillance is a dynamic Distributed Constraint Optimization Problem (DCOP). In dynamic DCOP a sequence of DCOPs will be solved at every discrete time step. For every discrete time step, the following three phases of the DPOP solver will be executed. First, using the constraint graph of the problem, the pseudo-tree is constructed. Afterwards, the utility propagation phase starts. Finally, as soon as the utility propagation phase is done, the value propagation phase begins.
This chapter will describe step-by-step how DPOP operates and how to use it for solving the problem of area surveillance. Furthermore, it will outline how the limitation of DPOP affects the performance of the solution.

4-1 Pseudo-tree construction

The DPOP operates over the pseudo-tree, hence, it starts by constructing the pseudo-tree arrangement from the constraint graph. The pseudo-tree is a re-arrangement of the constraint graph [11]. The constraint graph consists of nodes and edges, where the edges indicates the relations between the different nodes. For the area surveillance problem, the nodes represent

the mobile-sensors and the edges symbolize the utility constraints. Given the definition of the utility constraint, an edge between two nodes in the constraint graph exists if their active domains overlap. More precise, it exists if the two mobile-sensors have the same grid points in their active domains. Therefore, by defining the utility constraints between every mobile-sensor, the constraint graph can be defined. As explained in Chapter 2, based on the definition of the utility constraint, at every discrete time step several constraint graphs may exist. In this case, every constraint graph, is considered as a separate DCOP. In addition, using the constraint graph, every mobile-sensor can determine its neighbors since two mobile-sensors are neighbors if they share a utility constraint.
After the mobile-sensors have defined their neighbors, they should follow the DFS algorithm [30] (Algorithm 1) for defining the pseudo-tree arrangement. As it can be seen in Algorithm 1, the token procedure requires a mobile-sensor that will start this procedure, known as the root, and it also requires from every mobile-sensor to know its neighbors. A commonly used heuristic for defining the root of the pseudo-tree is to select the one with the most neighbors. Therefore, for the area surveillance problem, the root mobile-sensor will be the mobile-sensor that has the most neighbors. Moreover, as clarified above, every mobile-sensor define its neighbors using the constraint graph. Finally, the result of the DFS algorithm for every mobile-sensor is a label for all of its neighbors. More specifically, a neighbor can be classified in one of the following categories: parent, pseudo-parent, child, pseudo-child.
Let us take, as an example, the formation of the mobile-sensors given in Figure 4-1.

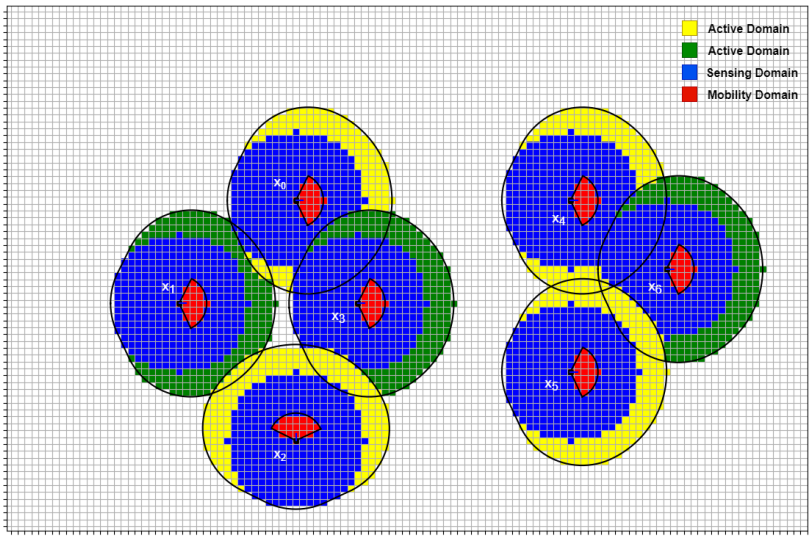


Figure 4-1: An example of 7 mobile-sensors where they create two constraint graphs. The green and yellow grid points represent the active domain, the red grid points correspond to the mobility domains and the blue grid points are the current sensing domains.

Obviously, in this formation, there are two distinct constraint graphs since there is no coupling between the two groups. Figure 4-2A shows the corresponding constraint graphs of the given example. As mentioned also before, for every constraint graph a root mobile-sensor should be picked. For both constraints graphs, there are several candidates since there are several mobile-sensors with the same amount of neighbors. In this case, the tie is broken using the identity of the mobile-sensor. To be more specific, the one that has the lowest identity will

be the root. Therefore, for the first constraint graph, the mobile-sensor α_0 is picked as the root and for the second constraint graph, the mobile-sensor α_4 is selected. After applying the DFS approach to both constraint graphs, the outcome, as can be seen in Figure 4-2B, is two pseudo-tree arrangements that do not have any relation between them. Therefore, both pseudo-trees have to solve separate their own area surveillance problem.

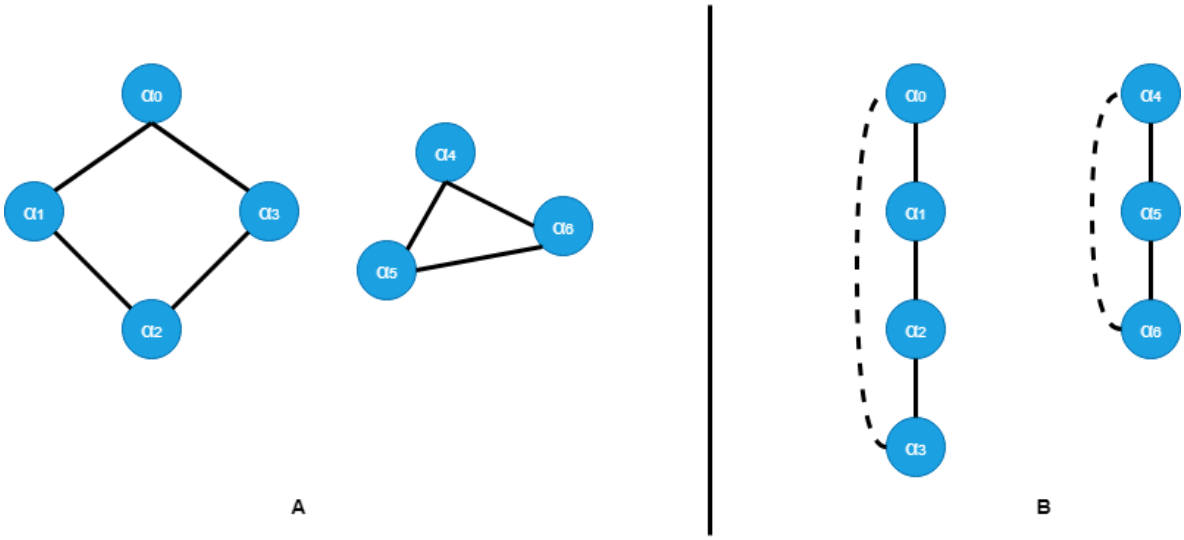


Figure 4-2: A) The constraint graphs of the formation shown in Figure 4-2. The edges (lines) represent the utility constraints and the nodes (circles) correspond to the agents. Two constraint graphs exist since there is no coupling between the two groups. B) The pseudo-trees of the constraint graphs shown in A. Solid lines indicate a parent-child relation and dashed lines show pseudo-parent - pseudo-child relation.

4-2 Utility propagation

After the pseudo-trees are constructed, the utility propagation phase starts. In this phase, the agents exchange utility messages, where the utility messages are utility matrices that contains utility values for variable assignments. The pseudo-trees act as a communication structure, where they specify for every agent to whom the agent has to send a utility message and from whom it has to receive utility messages. More specific, an agent has to send a utility message to its parent and it has to receive utility messages from its children. Moreover, an agent sends a utility message to its parent as soon as it has received every utility message from its children. The utility propagation phase starts from the leaves of the pseudo-trees, the agents that do not have any children, and it propagates up to the root [20]. The leaves start this procedure since they do not have any children and therefore, they do not have to wait for any utility message. Finally, the utility propagation phase ends as soon as the root of the pseudo-tree has received all of the utility messages from its children. This subsection will explain how the utility messages that the agents have to exchange are constructed.

4-2-1 Utility message

A utility message is a matrix of utility values. A utility message that is sent by agent α_κ to agent α_ρ is symbolized by U_κ^ρ and it consists of utility values u_κ^ρ. In the case that the agent α_κ does not have any pseudo-parents and its separator includes only its parent, the utility message will be a vector of utility values. Every utility value in the utility message represents the optimum utility $u_\kappa^\rho(d_\rho)$ that the sub-tree rooted at α_κ can achieve if agent α_ρ chooses the value assignment $d_\rho \in D_\rho$. Therefore, in this case, the utility message will contain as many utility values as the number of value assignments in the discrete domain of the parent agent α_ρ.
In the case that agent α_κ has also pseudo-parents or its separator includes more than one agent, the utility table will be a multi-dimensional matrix of utility values, where each utility value will be the optimum utility $u_\kappa^\rho(d_\rho, d_1, ..., d_n)$ that the sub-tree rooted at α_κ can achieve if agent α_ρ choose value $d_\rho \in D_\rho$ and the rest agents in its separator $\alpha_1, ..., \alpha_n \in s_\kappa$ choose value $d_1 \in D_1$, ... $d_n \in D_n$, respectively. Therefore, the utility message depends on the size of the separator of the sending agent. That is, the utility message will have an extra dimension for every agent in the separator of the agent α_κ.
For example, lets assume that agent α_0 has to send a utility message to its parent agent α_1, and the separator of agent α_0 includes agents α_1, α_2. Then the utility message U_0^1 will have one dimension for agent α_1 and one for agent α_2. If the domain of agents α_1 and α_2 are the following: $D_1 = \{1, 2\}$, $D_2 = \{3, 4\}$, then the utility message U_0^1 will be the following:

$$U_0^1 = \begin{array}{cc} & \alpha_1 \\ \alpha_2 & \begin{array}{c|c|c} & 1 & 2 \\ \hline 3 & u_0^1(1,3) & u_0^1(2,3) \\ \hline 4 & u_0^1(1,4) & u_0^1(2,4) \\ \hline \end{array} \end{array}$$

Compute the utility values

Utility messages are constructed by the following procedure. First, the agent has to compute the utility value for every possible combination of value assignments of its local variable and the local variables of its parent and/or pseudo-parent agents. These utility values will form a utility message between the sending agent and its parent. In addition, the sending agent has to incorporate the utility messages that it has received from its children, with its own utility message. To incorporate the utility messages, the sending agent has to sum up the utility values in the utility messages of its children with the corresponding utility values in its own utility message with its parent. Finally, the last step for the sending agent is to follow the projection operation. During the projection operation, the sending agent defines for every possible value assignment of the the local variables of the agents other than the sending agent, the optimal value assignment for its local variable. This will lead to define the maximum utility value that the sub-tree rooted at the sending agent can achieve for every possible value assignment of the local variables of the agents in its separator. After the projection operation is done, the utility message is ready to be sent to the parent agent.
In the following subsections, the procedure that the agents in the area surveillance problem follow for constructing the utility messages will be explained in more details.

4-2-2 Utility message for area surveillance

As explained above, a utility message consists of utility values for every possible combination of value assignments of the current agent and the agents in its separator. For the area surveillance problem, a value assignment of a mobile-sensor α_κ is a potential position in its mobility domain D_{M_κ}, where a potential position is any position that the mobile-sensor can reach within the next discrete time step based on its mobility constraints. Following the problem definition of the area surveillance problem in Chapter 1, a utility value depends on the following: the sum of probabilities of the grid points (f_c), known as coverage, the safety constraint function (f_s), and the long horizon function (f_h). Equation 4-1 shows how the utility value of the current mobile-sensor α_κ and its parent and pseudo-parents $\Psi_\kappa = \{\alpha_p, \alpha_{g_1}, ..., \alpha_{g_n}\}$ is defined for the given potential positions $(d_\kappa, d_p, d_{g_1}, ..., d_{g_n})$.

$$\begin{aligned} u^p_\kappa(d_\kappa, d_p, d_{g_1}, ..., d_{g_n}) &= f_c(d_\kappa, d_p, d_{g_1}, ..., d_{g_n}) + \\ &+ f_s(d_\kappa, d_p, d_{g_1}, ..., d_{g_n}) + \\ &+ f_h(d_\kappa) \end{aligned} \tag{4-1}$$

Coverage value The coverage value defines the sum of probabilities of the grid points that only the sending agent α_κ will have within its sensing range, if it chooses the potential position d_κ. Therefore, if any of its parent or pseudo-parents have the same grid points within their sensing range with the agent α_κ, these grid points have to be subtracted from the current coverage of agent α_κ. This is done to avoid counting any grid point twice during the utility propagation phase.
In order to compute the coverage value for the potential position d_κ of agent α_κ, first, the grid points that will be within its sensing range based on this position d_κ have to be defined. This is known as potential sensing domain (D_{Υ_κ}), and it can be defined using Equation 4-2

$$D_{\Upsilon_\kappa}(d_\kappa) := \{\bar{x}_i \in M \mid |d_\kappa - d_i|_2 \leq r_\kappa\} \tag{4-2}$$

where M contains every grid point in the environment and r_κ is the sensing range of agent α_κ. In addition, to define the common grid points that the agent α_κ will have with its parent and pseudo-parents, the potential sensing domains of its parent and pseudo-parents need to be defined. This can be done using Equation 4-2 for every potential positions $(d_p, d_{g_1}, ..., d_{g_n})$. Then, the combined potential sensing domain of the parent and pseudo-parents agents is computed using Equation 4-3.

$$S_\Psi = D_{\Upsilon_p}(d_p) \cup D_{\Upsilon_{g_1}}(d_{g_1}) \cup ... \cup D_{\Upsilon_{gn}}(d_{g_n}) \tag{4-3}$$

Finally, the coverage value between the mobile-sensor α_κ and its parent and pseudo-parents Ψ_κ can be defined using Equation 4-4.

$$f_c(d_\kappa, d_p, d_{g_1}, ..., d_{g_n}) = \sum_{\bar{x}_i \in D_{\Upsilon_\kappa}(d_\kappa)} P(\bar{x}_i) - \sum_{\bar{x}_i \in S_{\kappa,\Psi}} P(\bar{x}_i) \tag{4-4}$$

where $P(\bar{x}_i)$ is a function that returns the probability value of the grid point i, and

$$S_{\kappa,\Psi} = D_{\Upsilon_\kappa}(d_\kappa) \cap S_\Psi \tag{4-5}$$

Safety constraint: The safety constraint function is defined as shown in Equation 1-14. The safety constraint function $s_c(d_\kappa, d_\rho)$ computes a utility value based on the distance between the potential positions (d_κ, d_ρ) of the two mobile-sensors. Specifically, the safety constraint function returns a negative utility value if the distance between the potential positions of the two mobile-sensors is less than a pre-defined safety threshold. In the case that the current mobile-sensor α_κ has pseudo-parents, the combined utility is defined by Equation 4-6.

$$f_s(d_\kappa, d_p, d_{g_1}..., d_{g_n}) = s_c(d_\kappa, d_p) + s_c(d_\kappa, d_{g_1}) + ... + s_c(d_\kappa, d_{g_n}) \tag{4-6}$$

Long horizon: The long horizon function depends only on the current agent and the area within its active domain. The active domain of agent α_κ includes every grid point in the environment that the mobile-sensor α_κ can observe in the next discrete time step, based on its mobility and sensing domain. In the case that every grid point in its active domain has a probability value zero, the long horizon function is employed to guide the mobile-sensors towards its closest sub-area that still has unobserved grid points. A sub-area is a small region in the area under surveillance that can be defined by splitting the area under surveillance into smaller areas of the same size.
The long horizon function computes a utility value for the potential position d_κ that is defined by the distance between the potential position d_κ and the center's position of the closest sub-area (d_c) in the environment that has still unobserved grid points. The lower the distance is, the higher the utility value will be. The utility value of the long horizon function can be computed using Equation 4-7.

$$f_h(d_{\alpha_\kappa}) = \begin{cases} \frac{1}{|d_c - d_\kappa|} & \text{if } \sum_{\bar{x}_i \in D_{A_\kappa}} P(\bar{x}_i) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4-7}$$

4-2-3 Incorporate the utility messages

When a mobile-sensor α_κ receives a utility message from its child α_ρ, it incorporates it with its utility message by summing up the two utility messages. Lets assume that the separator of the child agent α_ρ includes only the receiver agent α_κ. Then, the child's utility message contains the utility value that the sub-tree rooted at agent α_ρ can achieve for every potential position $d_\kappa \in D_{M_\kappa}$ of the current agent α_κ. Hence, the agent α_κ sums the utility values of the two utility messages using Equation 4-8

$$u^p_\kappa(d_\kappa, d_p, d_{g_1}..., d_{g_n})' = u^p_\kappa(d_\kappa, d_p, d_{g_1}, ..., d_{g_n}) + u^\kappa_\rho(d_\kappa) \tag{4-8}$$

where the $u^\kappa_\rho(d_\kappa)$ is the utility message received from the child agent α_ρ. By incorporating the child's utility message with its own utility message, the resulting utility values represent the utility value that the sub-tree rooted at agent α_κ can achieve. However, in the case that the utility message from its children depends on other agents than the agent α_κ and the parents and pseudo-parents of agent α_κ, the agent α_κ should extend its utility message by including the missing agent. The mathematical expression for extending the dimension of a utility message can be seen in Equation 4-9 where the agent α_κ extends its utility message $u^p_\kappa(d_\kappa, d_p, d_{g_1}..., d_{g_n})$ by adding agent α_q. In this Equation it is assumed that the utility message from the child agent α_ρ depends on the agents α_κ and α_q.

$$u^p_\kappa(d_\kappa, d_p, d_{g_1}..., d_{g_n}, d_q)' = u^p_\kappa(d_\kappa, d_p, d_{g_1}..., d_{g_n}) + u^\kappa_\rho(d_\kappa, d_q) \tag{4-9}$$

This extension will increase the dimension of the utility message by one, and the number of values in the utility table by $|D_{M_q}|$ times.
Figure 4-3 shows how agent α_κ extends its utility message by adding agent α_q to its own utility message. Agent α_ρ sent to agent α_κ the utility message $U^\kappa_\rho = u^\kappa_\rho(d_\kappa, d_q)$ that its utility values depend on the agents α_κ and α_q. The values in the utility message of agent α_κ, $u^p_\kappa(d_\kappa, d_p)$ depends only on itself and its parent agent α_p. Therefore, the agent α_κ has to extend its utility message by adding an extra dimension for agent α_q. The resulting utility message $u^p_\kappa(d_\kappa, d_p, d_q)'$ is shown on the right.

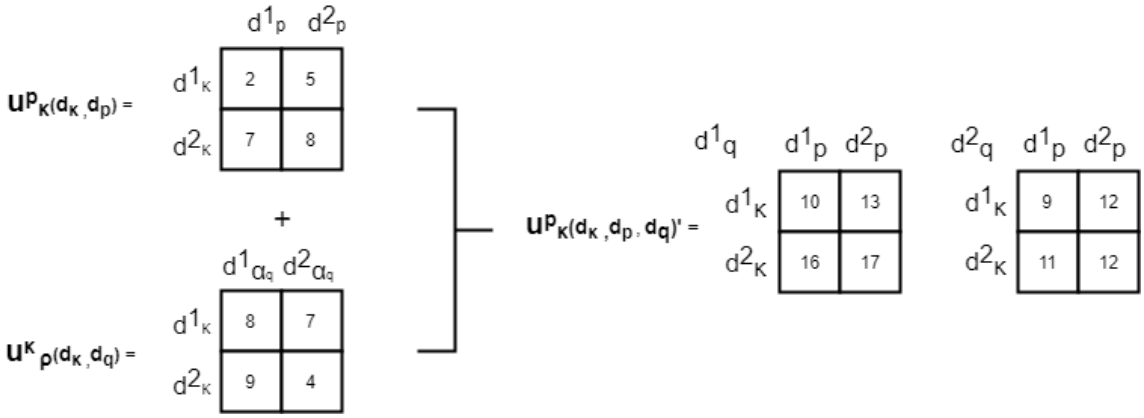


Figure 4-3: Combining the utility message u^p_κ of agent α_κ with the utility message u^κ_ρ from its child agent α_ρ. The resulting utility message is shown on the right where the agent α_q has been added to the utility message u^p_κ.

4-2-4 Utility message for parent

By incorporating the utility messages from its children in its utility message, the agent α_κ has the optimal utilities that the sub-tree (rooted at itself) can achieve. The final step before sending the utility message to its parent is to define for every combination of potential positions of the agents in its separator, its potential position that results to the maximum utility. This can be done by following the projection operation ($*$). The projection operation is defined as shown in Equation 4-10.

$$u^p_\kappa(d_p, d_{g_1}, ..., d_{g_n}) = \max_{d_\kappa \in D_{M_\kappa}} u^p_\kappa(d_\kappa, d_p, d_{g_1}, ..., d_{g_n})' \tag{4-10}$$

The outcome of the projection operation is the utility message that the agent α_κ will send to its parent agent α_p, $U^p_\kappa = u^p_\kappa(d_p, d_{g_1}..., d_{g_n})$. For example, based on the utility messages presented in Figure 4-3, after the agent α_κ followed the projection operation, the utility message that the agent α_κ will send to its parent agent α_p will be the following:

$u^p_\kappa(d_p, d_q) =$

	d^1_q	d^2_q
d^1_p	16	11
d^2_p	17	12

4-2-5 End of utility propagation phase

The root of the pseudo-tree is the last agent that will receive the utility messages from its children. As soon as it has received every utility message, it first sums all the utility messages together using Equation 4-11.

$$u_r(d_r) = \sum_{\rho \in C(\alpha_r)} u^r_\rho(d_r) \tag{4-11}$$

The root agent α_r has now full knowledge of what is the maximum utility that the rest of the agents in the pseudo-tree can achieve for every possible value assignment of itself. Furthermore, the root has to define for every potential position in its mobility domain $d_r \in D_{M_r}$ its own local coverage value. Since it does not have any parent and pseudo-parents, the local coverage value for the agent root can be computed using Equation 4-12.

$$f_c(d_r) = \sum_{\bar{x}_i \in D_{R_r}} p(\bar{x}_i) \tag{4-12}$$

The root of the pseudo-tree computes the local coverage value since all of the utility constraints have been incorporated in the utility messages of its children. Finally, the utility value for the potential position d_r of the root agent α_r is computed using Equation 4-13.

$$u_r(d_r)' = u_r(d_r) + f_c(d_r) + f_h(d_r) \tag{4-13}$$

This utility value represents the global utility that the agents in the pseudo-tree can achieve if the root agent chooses the value assignment d_r.

4-3 Value propagation

The value propagation phase starts as soon as the root of the pseudo-tree has received every utility message from its children. In this phase, every mobile-sensor is able to define its optimal position. To do so, the root of the pseudo-tree starts by computing its position assignment that based on its utility message $u_r(d_r)'$ results to the maximum utility value. The root can define its optimal position d^*_r using Equation 4-14.

$$d^*_r = \arg\max_{d_r \in D_{M_r}} u_r(d_r)' \tag{4-14}$$

In addition, it creates the value message and appends its optimal position $V_r^{C(\alpha_r)} = \{d^*_r\}$. The value message $V_r^{C(\alpha_r)}$ is sent by agent α_r to its children agents $C(\alpha_r)$, and it contains the optimal value assignment of the local variables of the root agent. Then the root agent sends this value message to its children. As soon as the children of the root agent receive the value message, they can compute their own optimal position using the value assignment of the root agent. Accordingly, given that the optimal position of its parent is d^*_r and by using their utility matrix $u^r_{C(\alpha_r)}$, they first define their own optimum position and then they append it to the value message $V_c^{C(c)} = \{d^*_r, d^*_{C(\alpha_r)}\}$. Finally, they send the value message to their own children. This procedure is repeated until the leaves of the pseudo-tree have received

the value message from their parent. It is important to mention that an agent requires the value assignment of every mobile-sensor that is included in its separator to compute its own optimum value. For example, if $s_\rho = \{\alpha_\kappa, \alpha_r\}$ then the agent α_ρ can define its optimal position using Equation 4-15 and the optimal positions of both agents α_κ and α_r.

$$d_\rho^* = \underset{d_\rho \in D_{M_\rho}}{\arg\max}\ u_\rho^\kappa(d_\rho, d_\kappa^*, d_r^*)' \tag{4-15}$$

4-4 Example of area surveillance problem

Figure 4-4 shows an example of the area surveillance problem with three mobile-sensors. The mobile-sensor α_0 is located at $d_0 = (5, 3)$, the mobile-sensor α_1 at $d_1 = (8, 6)$, and the α_2 at $d_2 = (3, 5)$. The sensing range of every mobile-sensor is shown with the orange circle and is equal to 1 grid point. Moreover, the heading of every platform is given with the blue arrow. In addition, the mobility range is represented with the red sector. Therefore, the mobility domain of mobile-sensor α_0 is $D_{M_0} = \{(5, 3), (4, 4), (5, 4), (6, 4)\}$. Furthermore, the active domains are depicted with red, blue and yellow colors for the mobile-sensors α_0, α_1, and α_2, respectively.
Based on the active domains of the mobile-sensors, agent α_0 shares a utility constraint with the agents α_1 and α_2. These three mobile-sensors aim to solve the area surveillance problem

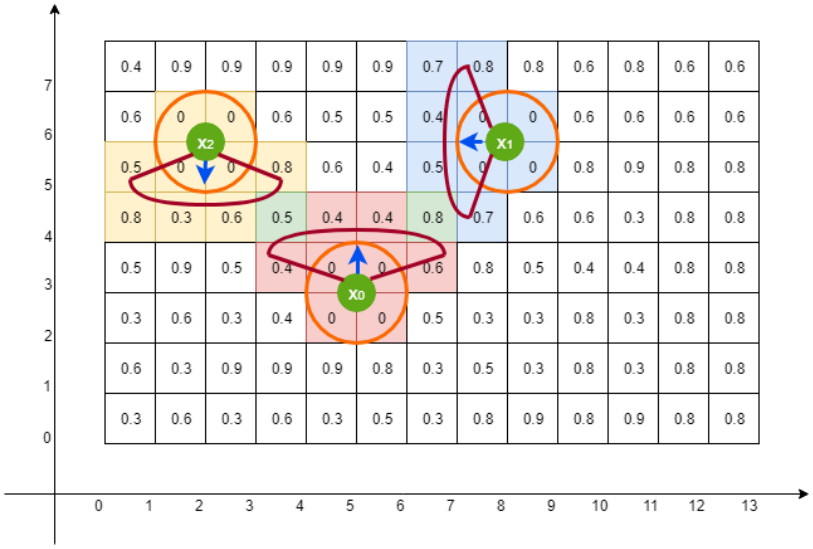


Figure 4-4: An area surveillance example with three mobile-sensors where the number in the grid points represent the likelihood that an object exist in the corresponding gird point. In this example, for every mobile-sensor, the sensing range is shown with the orange circle, the heading with the blue arrow, and the mobility-range with the red sector. The red, blue and yellow grid-points represent the active domains of mobile-sensors α_0, α_1 and α_2, respectively. Moreover, the green grid points represent the grid points that the active domain of mobile-sensor α_0 has in common with the active domains of mobile-sensors α_1 and α_2.

using the DPOP solver. Therefore, at this discrete time step, the three-mobile sensors have to coordinate their actions in order to define their next position that results in the maximum

coverage. Figure 4-5 shows the messages that the three agents exchange during the utility and value propagation phase. Agents α_1 and α_2 start by computing their utility message following the utility propagation phase explained in this Section 4-2. After applying the projection operation, they send the utility message to their parent agent α_0. Then, agent α_0 defines its own local utility message using Equation 4-13 and afterwards it incorporates the utility messages with its local utility message. In addition, it defines its position that results to the maximum utility value, $V_0^{\{\alpha_1,\alpha_2\}} = \{d_0^* = d_0^4\}$, and it sends it to its children through a value message. The agents α_1 and α_2, using the optimal position of their parent, can determine their own optimal position. Finally, the optimal position for every mobile-sensor was found to be $d_0^* = (6,4)$, $d_1^* = (7,7)$, $d_2^* = (1,6)$.

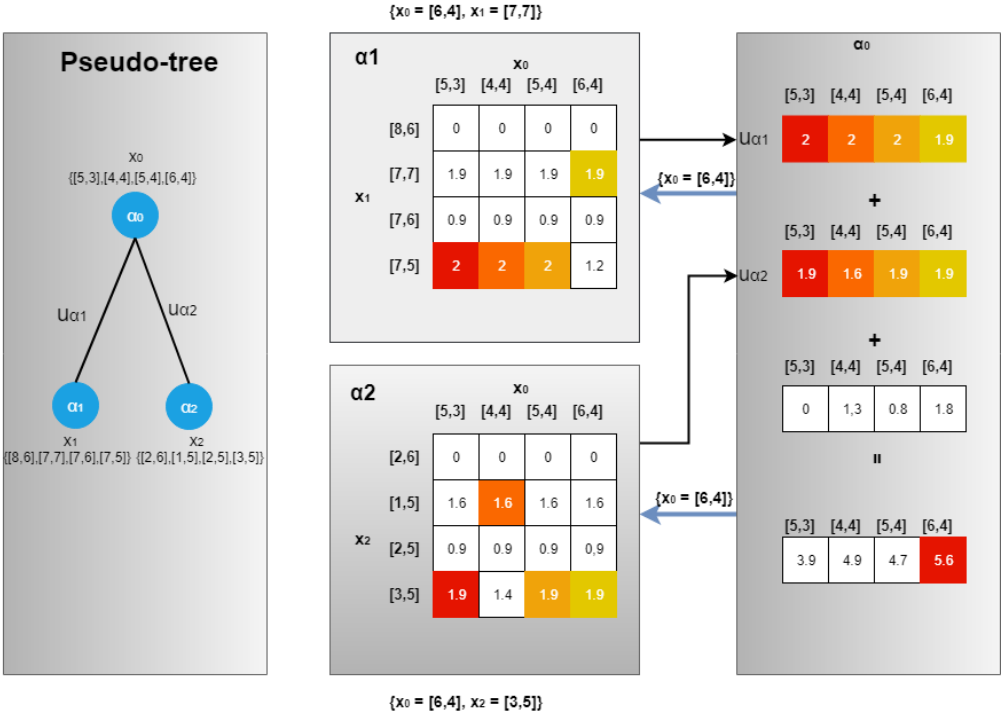


Figure 4-5: On the left side the pseudo-tree of the DCOP problem can be seen. On the right side, the message flow during the DPOP is depicted. First, agents α_1 and α_2 constructs their utility messages using the utility propagation phase (Section 4-2). Then, they follow the projection operation ($*$) where they define the optimum utility for every value assignment of their parent(shaded colors). Finally, they send the utility message to their parent agent α_0. Agent α_0, after receiving the utility messages, determines its optimal position and it sends it as a value message to its children. Finally, agents α_1 and α_2 compute their optimal position using the value message of agent α_0

4-5 Memory requirements

The DPOP is a complete solver that finds the optimal solution, through exchanging a linear number of messages with respect to the number of agents [30, 20]. Nevertheless, the size of these messages depends on the induced width of the pseudo-tree. The induced width is equal to the size of the largest separator of any agent in the pseudo-tree [30]. Specifically,

the largest message that the mobile-sensors have to exchange will be space exponential in the induced width of the pseudo-tree. Consequently, DPOP demands a large amount of memory to store the data, whenever the induced width of the pseudo-tree is high.
When the agents have a limited amount of storage, this drawback limits the number of agents that can be utilized for solving the DCOP. Considering the area surveillance problem, the mobile-sensors are equipped with onboard computers that have a limited amount of available memory. Therefore, if the induced width of the pseudo-tree is high, some of the mobile-sensors will not be able to construct their utility messages. For a better understanding of how does this drawback limits the number of mobile-sensors, the size requirements will be elaborated.

Size of a utility message

The utility message for a mobile-sensor contains the utility value for every possible combination of positions of the current mobile-sensor and the mobile-sensors in its separator. Therefore, the number of values that the utility matrix has depends on the size of its separator, the number of variables per agent within the separator, the size of the mobility domain for every mobile-sensor in its separator, and the size of its own mobility domain. The number of values that the utility matrix of mobile-sensor κ is given by Equation 4-16.

$$v_\kappa = |D_{M_\kappa}| \times \prod_{q \in s_\kappa} |D_{M_q}| \tag{4-16}$$

Furthermore, the required storage size of the utility message can be computed by multiplying the required memory to store a single value with the total number of values that the utility message has.
An example of a pseudo-tree with a high induced width can be seen in Figure 4-6. In this pseudo-tree, 7 agents exist where every agent has a utility constraint with every other agent. This will create a pseudo-tree with an induced width of 6. If every mobile-sensor has the same size of the mobility domain, $\eta = |D_{M_\kappa}|$, then the utility matrix of mobile-sensor κ will consist of $\eta^{|s_\kappa|+1}$ values.
Based on the same pseudo-tree example, if the size of the mobility domain is larger than 20 and by assuming that a single value requires 8 bytes of memory, then agent 7 will require more than $8Gb$ of memory to construct and store the utility matrix. Consequently, if the agent has less than $8Gb$ of available memory, the agent 7 will not be able to construct the utility matrix and the DPOP will fail to solve the problem.

4-6 Conclusion

In this chapter, the DPOP solver was further explained. At every discrete time step, every agent within the pseudo-trees has to follow the utility and value propagation phase that have been described in this chapter. By doing this, every mobile-sensor is able to compute its position that will result in the maximum coverage for the next discrete time step. Using the DPOP solver, defining the optimal postion for every mobile-sensor can be achieved by using a linear number of messages. However, the main limitation of the DPOP solver makes it inappropriate for solving the area surveillance problem. More specifically, if the induced width of any of the pseudo-trees is high, then the mobile-sensors will require a large amount

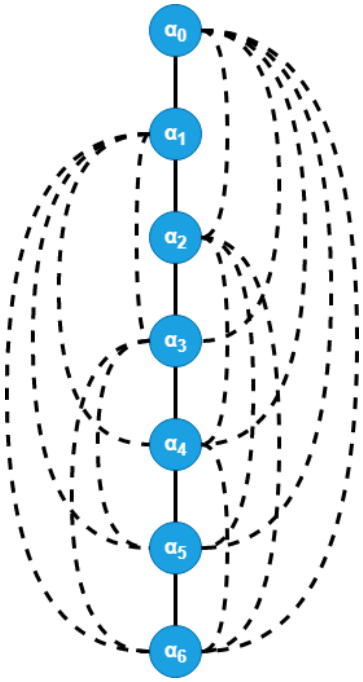


Figure 4-6: An instance of a pseudo-tree with high induced width. The agents are the circles where two agents are connected with an edge (line) if they share a utility constraint. The solid lines represent a parent-child relationship and the dashed lines indicate a pseudo-parent - pseudo-child relation. The induced width of this pseudo-tree is equal to 6 since the separator of agent α_6 consists of the agents $\{\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$.

of memory to store the utility matrices. If the mobile-sensor do not have enough memory for constructing the utility matrices, the agent will not be able send the utility message to its parent and the utility propagation phase will terminate. Consequently, due to the fact that mobile-sensors have a limited amount of available memory, there is a limit on the number of mobile-sensors that can be employed for the area surveillance problem. Hence, DPOP does not satisfy the solver's requirement of being scalable with respect to the number of mobile-sensors.

In order to overcome this limitation, an extension of DPOP that can successfully overcome the high demand for memory requirement will be designed. Based on the available extensions of DPOP, there are several ways to overcome this memory requirement. The next chapter will present some of the available extensions of DPOP and it will introduce a new extension in order to overcome the high demand of memory.

Chapter 5

DPOP extension

The Distributed Pseudo-tree Optimization Procedure (DPOP) is an inference-based solver that checks the whole search space in order to determine the optimal solution. One of its key limitations is the size of the largest message that the agents have to exchange during the utility propagation phase. Moreover, as shown in the previous chapter, this limitation restricts the number of mobile-sensors that can be employed for solving the area surveillance problem.
Since the DPOP will be utilized for solving the area surveillance problem, a solution for this problem needs to be found. One of the solutions is to optimize the induced width of the pseudo-tree. However, optimizing the induced width of the pseudo-tree is an NP-hard problem and solving an NP-hard problem can introduce extra computational complexity [38]. Furthermore, optimizing the induced width of the pseudo-tree cannot guarantee that the resulting pseudo-tree will be of low induced width. Hence, a different approach is required.
This chapter will first give an overview of the available extensions of the DPOP solver which focus on how to reduce the memory requirements. In addition, this chapter will introduce a new algorithm, an extension for the DPOP, which has been designed based on the area surveillance problem. The main goal of the new extension is to reduce the memory requirements of the DPOP so it can be efficiently applied to a real-world scenario. Eventually, this chapter will prove that this extension can provide an error bound to the solution of the problem. That is, a bound on the error between the solution of the solver and the global optimum.

5-1 Available extensions of DPOP

The exponential memory requirements of the DPOP restricts the use of the DPOP for real-world multi-agent systems. This is due to the fact that in multi-agent systems the agents are equipped with onboard computers with limited computational resources. Therefore, if the agents have to construct and store utility messages that are higher than their available memory, they will not be able to define the solution to the problem. During the last few years, the DPOP has been extended in various ways by several researchers where their main

focus was to reduce the search space for the optimal solution.
The majority of the researchers focus on how to reduce the domains of the agents so they can decrease the search space and the size of the message. Ismel and Pedro [39] introduce the use of function filtering for reducing the size of the largest message. Fioretto et al [35] make use of branch consistency for reducing the size of the messages. More specific, using the hard constraints of the problem they were able to prune the search space. Based on the same idea, H-DPOP [36] exploits hard constraints for smaller runtimes (number of operations executed for defining the solution). However, compared to the branch consistency, H-DPOP requires longer runtimes to define the solution, where also DPOP with branch consistency can scale larger problems than H-DPOP [35].
Besides the use of hard constraints, some other researchers overcomes this limitation using memory restrictions. To be more specific, O-DPOP and MB-DPOP trade off memory requirement for longer runtimes [40, 37]. Based on the same idea, Petcu introduced the A-DPOP [41] solver which trades off solution optimality for shorter runtimes.
Furthermore, other researchers [33, 42] recommended replacing the pseudo-tree arrangement that the conventional DPOP utilizes. Chen et al [33] proved that by using the Breadth First Search (BFS) for deriving the pseudo-tree arrangement, the runtime can be reduced. Using the BFS algorithm they achieved to slightly decrease the maximal dimension of the utility message. This is because BFS result in much more branches than DFS. Moreover, BFS can result in a pseudo-tree with lower induced width. In addition, Vinyals et al [42] use junk trees to solve the DCOP problem. After a comparison with DPOP, they concluded that their algorithm outperforms DPOP in terms of computation, communication, and parallelism.
Consequently, there are three different approaches to reduce the size of the message. The first approach is to use the hard constraints of the problem to prune the domain of the agents [39, 35, 36]. Using the hard constraints for pruning the domains might lead to reducing the size of the message, however, it highly depends on the hard constraints of the problem, and whether the hard constraints provide sufficient information for pruning the domains of the agents. Moreover, the second approach is to apply hybrid algorithms to overcome the memory restrictions. That is, to utilize the conventional DPOP whenever there is memory available and a bounded DPOP whenever there is no memory available [37]. By utilizing the hybrid DPOP can result in reducing the size of the message, however, this approach requires longer runtimes to compute the solution and this is not desirable for the area surveillance problem. Finally, the last approach is to change the pseudo-tree arrangement by either using another approach to derive the pseudo-tree or by employing a different representation [33]. This approach may result in reducing the induced width of the pseudo-tree, however, it slightly decreases the size of the messages. This implies that the memory requirements of the utility messages will still be high and the agents will still be unable to construct the utility messages. Therefore, such an approach will not make the DPOP applicable to the area surveillance problem.
To conclude, based on this section and based on the available extensions in the literature, only the MB-DPOP can successfully overcome the high memory requirements of DPOP and at the same time guarantee the global optimum solution [37]. To achieve this, the MB-DPOP increases the number of messages that the agents have to exchange, and as explained in the previous chapter this is not desired. Moreover, the MB-DPOP requires longer runtimes to define the solution compared to the DPOP. Given the fact that the solver should be able to run during the surveillance of the area, it is not desirable to have longer runtimes. Therefore, a new technique has to be designed. This new technique will combine the idea of both the first

and the second approach. That is, a new bounded-DPOP will be designed and will be utilized whenever the induced width of the pseudo-tree is high. Moreover, the bounded-DPOP will make use of the utility constraints to reduce the size of the utility messages.

5-2 Insight of the new extension

During the surveillance of the known area, it could be that all the mobile-sensors are close to each other. Based on their active domains, this can result in an area surveillance problem where every agent shares a utility constraint with every other agent. This creates a constraint graph with $\frac{(N-1)*N}{2}$ edges, where N is the number of agents. In addition, by utilizing the DFS algorithm for deriving the pseudo-tree, it will result in a pseudo-tree with an induced width of $N-1$. Hence, the dimension of the biggest message that the agents will exchange during the utility propagation phase will be $N-1$. In order to avoid storing and constructing such a big utility messages, action needs to be taken whenever the induced width of the pseudo-tree is high.
Based on the approach in MB-DPOP [37], the size of the message can be reduced by removing areas in the pseudo-tree that have high induced width. More simply, by removing edges and nodes from the pseudo-tree that increases the induced width. Moreover, based on the approach in [35], the size of the messages can be reduced by utilizing the hard constraints of the problem for pruning the domains of the agents. Consequently, to get the most out of both approaches, the new extension will be based on both of them, where edges that increases the induced width of the pseudo-tree and does not have big impact on the optimal solution will be removed from the DCOP to reduce the size of the utility messages.
Considering the area surveillance problem, only the safety constraint can be considered as hard constraint. However, it cannot be utilized for reducing the size of the messages since it does not provide enough information for the optimal solution. Besides the safety constraint, in the area surveillance problem the agents share utility constraints. A utility constraint or an edge in the area surveillance problem exists if the active domains of two mobile-sensors contain the same grid points of the environment. In this case, the mobile-sensors should exchange information for constructing the utility messages. The information that these two mobile-sensors will exchange depends on the probability value of the grid points that the active domains of these two mobile-sensors have in common. Based on that, some edges do not share any important information for the optimal solution since the grid points that the mobile-sensors have in common have low probability value compared to other edges. Hence, these edges do not have a large influence on the optimal solution. The solution is to remove these edges from the problem to avoid any unnecessary computation, communication and memory requirements. Removing an edge from the problem means that the corresponding mobile-sensors will not have to exchange messages for computing the solution nor to consider the other mobile-sensor for constructing the utility message during the utility propagation phase. Hence, the number of values in the utility messages is reduced.
Nevertheless, it is challenging to determine how many edges and which edges have to be removed. For this reason, an algorithm that aims to get rid of the edges that have low impact on the optimal solution will be utilized. One of the most commonly used techniques for removing edges in the pseudo-tree is the Maximum Spanning Tree (MST) algorithm [43, 44]. This approach requires a weighted constraint graph or pseudo-tree, where based on the weights of the edges it defines the edges that have to be removed. The remainder of this chapter will

explain in more details the MST algorithm and how it can be utilized as an extension of the DPOP.

5-3 Maximum spanning tree

A spanning tree is a tree that consists of a subset of edges of a constraint graph that connects all the nodes in the tree without creating cycles [43, 44, 45]. Assuming that the edges have a value or weight that indicates the importance of the edge for the problem, then the maximum spanning tree is the sub-tree where its sum of the edges results in the maximum value/weight. Two of the most common algorithms for deriving the maximum spanning tree are the Kruskal's algorithm [46] and the Prim's algorithm [47]. Their outcome is the same, however, they differ in the procedure for constructing the MST.

Kruskal's algorithm

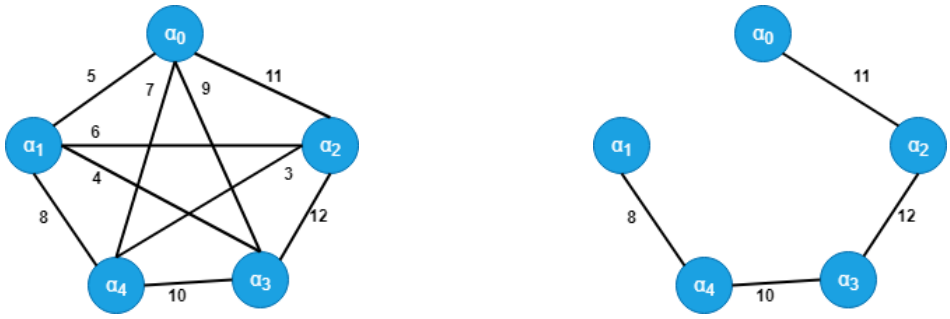
Kruskal's algorithm tries to define the maximum spanning tree by finding the edge with the highest weight that connects two different trees [46]. Initially, every node in the constraint graph forms a tree. Then, it determines the edge with the highest weight. If this edge connects two nodes that belong to two different trees, it combines the trees. Otherwise, it skips the edge. This approach is repeated until every edge has been explored. In the end, this algorithm will result in the maximum spanning tree that connects all of the nodes without creating any cycles.

Prim's algorithm

Prim's algorithm, on the other hand, builds the maximum spanning tree by iteratively adding the node that results to the highest weight [47]. Initially, it randomly chooses a node. Additionally, it finds the node that is connected to the initial node, and it has the maximum edge weight. This procedure continues by adding every time the node that is connected with the tree, and it has the maximum weight. Again, it skips the edge if it creates a cycle.

MST example

As the definition of the MST indicates, the outcome will be a sub-tree of the constraint graph, where it does not create any cycles and it has the maximum sum of weights. Such an example can be seen in Figure 5-1. Figure 5-1a shows a constraint graph that consists of 5 agents and 10 edges along with their weights. After applying the Kruskal's or Prim's algorithm, the MST shown in Figure 5-1b is constructed. This MST consists of 5 agents and 4 edges.



(a) The initial constraint-graph with 5 agents and 10 edges.

(b) The resulting MST after applying the Kruskal's algorithm.

Figure 5-1: Apply the MST algorithm to the constraint graph in (a). The MST of the constraint graph is depicted in (b)

5-4 Apply MST to the area surveillance problem

An MST algorithm requires a weighted constraint graph or pseudo-tree for determining the MST. Consequently, before applying the MST algorithm to the area surveillance problem, the weight of each edge has to be defined. This procedure is known as the weighting procedure and is explained below.

5-4-1 Weighting procedure

The MST approach defines the spanning tree that results to the maximum sum of weights. Consequently, any edge with low weight will not be part of the final spanning tree. Considering the area surveillance problem, an edge between two agents means there is a utility constraint between them, and therefore the two agents could result in covering the same grid points. From the definition of the utility constraint, an edge can take its weight using the grid points that the active domains of the two agents have in common. Every grid point has its own probability value. Therefore, the sum of the probabilities of these grid points can set up the weight for each edge. Using this technique for defining the weight for the edges, the weight can define the influence that the edge will have on the global utility function. Equation 5-1 shows how to define the weight for an edge that connects agent α_κ and agent α_ρ

$$w_{\kappa,\rho} = \sum_{\bar{x}_i \in D_{S_{\kappa\rho}}} P(\bar{x}_i) \tag{5-1}$$

where $P(\bar{x}_i)$ is a function that retuns the probability value of the grip point i and the $D_{S_{\kappa\rho}}$ is the share domain and it consists of the grid points that active domains of agents α_κ and α_ρ have in common. Therefore, if an edge in the constraint graph has a low weight, it indicates that either the share domain of the agents contains only a few grid points or that the grid points that are included in the share domain have low probability value. In both cases, the information that the two agents will exchange during the utility propagation phase will have a low impact on the optimum solution. Hence, it is preferred to ignore the edges with a low weight in order to keep the error of the solution as low as possible.

Tie-breaker

When the MST algorithm is applied to the constraint graph of the area-surveillance problem, it could be that two or more edges have equal weights. In this case, for each edge, the current positions of the two agents that are part of the edge are utilized to break the tie. To be more specific, the actual distance in euclidean space between the positions of the two agents will be considered. Since these edges have the same utility relation based on the sum of the probability values, it is preferable to keep in the DCOP the edge that its agents are closer to each other. This will allow the two agents to communicate during the utility and value propagation phase so they will avoid any collision. Therefore, whenever there is a tie between edges, the inverse of the distance will be added to the weight of the edges as shown in Equation 5-2.

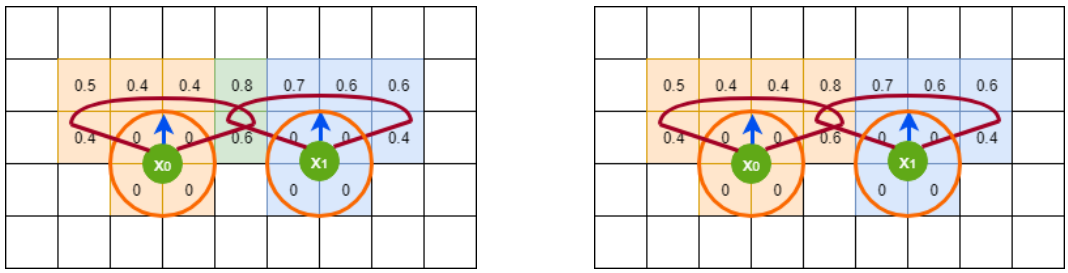
$$w'_{\kappa,\rho} = w_{\kappa,\rho} + \frac{1}{|d_\kappa - d_\rho|_2} \tag{5-2}$$

After defining the weights for each edge in the DCOP and by applying any of the aforementioned MST algorithms, the outcome will be a set of edges that have to be removed from the DCOP. By removing them from the DCOP, the outcome will be a constraint graph with N-1 edges. This is desired since the pseudo-tree that will be constructed from this constraint graph will have at maximum an induced width of 1. Hence, the largest utility message will be reduced and its maximum dimension will be 1.
Nonetheless, removing an edge from the DCOP implies that the two mobile-sensors will not coordinate their actions for defining the optimum solution. Furthermore, since the two mobile-sensors share a utility constraint but they will not communicate for defining the optimum solution, it is possible that they will end up in covering the same grid points of the environment. Based on the definition of the area-surveillance problem, it is not desirable to have two mobile-sensors sensing the same grid points. Consequently, to avoid this, an extra step known as the edge elimination procedure, will be utilized.

5-4-2 Edge elimination procedure

When an edge between two mobile-sensors is removed from the DCOP, it is important that these two mobile-sensors will not observe the same unobserved grid points in the next discrete time step. In order to achieve this, the share domain and the active domain of the mobile-sensors will be utilized. Specifically, the grid points that exist in the share domain will be blocked from the active domain of any of these two mobile-sensors. Blocking a grid point from the active domain of any mobile-sensor means to remove the grid point from the active domain of the mobile-sensor. Hence, the mobile-sensor will not get any utility gain if it has within its sensing range the blocked grid point. So, it will not be possible any more for both to end up in covering the same grid points.
This can be better understood with an example. Figure 5-2a shows two agents with their active domains (orange color agent α_0, blue color agent α_1) and their share domain (green color). To eliminate the edge between this two agents, the grid points within the share domain should be blocked from the active domain of one of the two agents. Let us assume that the share domain will be blocked from the active domain of agent α_1. This will result in Figure

5-2b. Therefore, if any potential position of agent α_1 result in having within its sensing domain the blocked grid points, instead of getting 1.4 utility gain, it will get 0. Hence, even without communicating during the utility and value propagation phase, the two agents will not choose the potential position that result in covering the same grid points.



(a) Before edge elimination. **(b)** After edge elimination.

Figure 5-2: The effect of the edge elimination procedure. Orange and blue colors represent the active domains of the agents and green color represents the shared utility. After the edge elimination procedure, the active domain of the mobile-sensor α_2 does not include the grid points of the share domain.

Reduce the active domain of the agents

Based on the edge elimination procedure, the share domain $D_{S_{\kappa\rho}}$ of the edge between the agents α_κ and α_ρ will be used for reducing the active domains of the agents α_κ and α_ρ. To choose from which agent to remove the corresponding grid point i from its active domain, the distance in euclidean space between the position of the center of the grid point i and the current positions of the mobile-sensors α_ρ and α_κ needs to be computed. Equation 5-3 shows how to compute the euclidean distance between the position of the center of the grid point i and the current position of the mobile-sensor α_κ.

$$e_{i\kappa} = \sqrt{(x_i - x_\kappa)^2 + (y_i - y_\kappa)^2} \tag{5-3}$$

Finally, based on the distance of both agents, the one that is further away from the grid point will have to remove the corresponding grid point from its active domain. In the case that the distances of both mobile-sensors are equal, the mobile-sensor with the higher sum of probabilities in its active domain will have to remove the grid point from its active domain. Equation 5-4 shows the updated domain of agent α_κ after removing the grid point i from its active domain.

$$D'_{A_\kappa} = D_{A_\kappa} / \bar{x}_i \tag{5-4}$$

This procedure is repeated for every grid point in the share domain $D_{S_{\kappa\rho}}$.

Eliminate all the edges

The MST approach returns a sub-set of edges that are utilized for reducing the active domains of the mobile-sensors. When more than one edges need to be eliminated, an extra step is required. This extra step is to update the remaining edges in case they have the same grid

point with the grid point that has just been removed from the active domain of the mobile-sensor. In more detail, when an agent α_κ removes a grid point from its active domain, it checks whether any other edge that includes itself has the blocked grid points $\bar{x}_i$ within its share domain. If this is the case, the agent α_κ updates the share domain using Equation 5-5

$$D'_{S_{\kappa\rho}} = D_{S_{\kappa\rho}} / \bar{x}_i \tag{5-5}$$

This process is repeated for every grid point that has been blocked. By applying this procedure for every grid point, it ensures that a grid point will not be removed from the active domain of two separate agents. The edge elimination procedure ends when all of the edges that have been rejected from the MST approach, have been used for blocking the grid points from the active domains of the agents.
The MST approach and the edge elimination procedure will be used as an extension of the DPOP. The DPOP along with this extension is known as MST-DPOP.

5-5 MST-DPOP

The DPOP solves the area surveillance problem in a decentralized manner and it consists of three phases. The MST-DPOP is an extension of DPOP and therefore it inherits these three phases. However, the MST-DPOP requires an extra phase for defining the edges that have to be eliminated from the DCOP. More precisely, after the pseudo-tree has been constructed and before applying the utility propagation procedure, MST-DPOP has to follow the edge elimination procedure explained in the previous sub-section.
To perform the edge elimination procedure in a decentralized manner, the agents first have to propagate edge-detail messages from the leaves to the root and then edge-reduction messages from the root to the leaves.
An edge-detail message is sent by the agent to its parent and it consists of the following:

$$M_{ed_\kappa} := \{E_\kappa, W_{DS_\kappa}, D_{S_\kappa}, W_{A_\kappa}, d_\kappa\}.$$

More details regarding the information that is included in the edge-detail message is given below.

- E_κ: The edges that the agent α_κ has. For example, if agent α_3 has an edge with agent α_1 and agent α_2 then $E_3 = \{(1,3),(2,3)\}$.
- W_{DS_κ}: The weight for every edge of agent α_κ. For example, $W_{DS_3} = \{w_{1,3} = 10, w_{2,3} = 12\}$.
- D_{S_κ}: The share domain for every edge of agent α_κ. For example, $D_{S_3} = \{D_{S_{13}}, D_{S_{23}}\}$.
- W_{A_κ} : The sum of probabilities of the active domains of the agents that agent κ share a utility constraint. For example, $W_{A_3} = \{w_{A_3} = 20,\ w_{A_2} = 12,\ w_{A_1} = 22\}$ where $w_{A_3} = \sum_{\bar{x}_i \in D_{A_3}} P(\bar{x}_i)$.
- d_κ: The current position of agent α_κ, $d_\kappa = (x_\kappa, y_\kappa)$.

An agent sends its edge-detail message to its parent if it has received all of the edge-detail messages from its children. When an agent receives an edge-detail message from its child c_1, it combines the information within the message with its own edge-detail message,

$$M_{ed_\kappa} := \{E_\kappa \cup E_{c_1}, W_{DS_\kappa} \cup W_{DS_{c_1}}, D_{S_\kappa} \cup D_{S_{c_1}}, W_{A_\kappa} \cup W_{A_{c_1}}, [d_\kappa, d_{c_1}]\}.$$

If an agent has several children, it does this for all children. Furthermore, when the root has received every edge-detail message from its children, it first combines all the edge-detail messages together, M_{ed_r}, and then it applies the MST algorithm and the edge elimination procedure. The root, using the information that is included in the edge-detail message (M_{ed_r}) starts by applying the MST approach. Specifically, using the E_r, the W_{DS_r} and the Kruskal's algorithm, the root defines the edges that have to be removed. To be more specific, using the Kruskal's algorithm it creates the new set C and it stores the edges that have to be removed in this set. That is, initially the tree consists of just the node of the root. Then, it iteratively finds the edge with the maximum weight and if it does not create any cycle it adds it to the tree. Otherwise it adds the edge to the set C.
Furthermore, using the set C, the current positions of the agents in its edge-detail message, the share domains D_{S_r}, and the edge elimination procedure, the root is able to define whether an agent has to remove grid points from its active domain. In this phase, the root creates for every agent in the pseudo-tree a set D_{BP_κ} that consists of the grid points that the agent α_κ has to block from its active domain. To do so, it starts from the first edge that is included in the set C, for example (κ, ρ), where (κ, ρ) is the edge between agents α_κ and α_ρ. For every grid point $\bar{x}_i$ that is included in the share domain of the edge (κ, ρ), the root agent follows the edge elimination procedure explained before. Assuming that the agent α_κ has the highest distance, then the root will store this grid point to the set $D'_{BP_\kappa} = D_{BP_\kappa} \cup \bar{x}_i$. In addition, it will update the remaining edges as explained before. Then it will move to the next edge in the set C. This process is repeated until every edge in the set C has been examined.
In the end, the root will have for every agent in the pseudo-tree, the grid points that they have to block from their active domain. Hence, the root creates the reduction message M_{μ_r} that consists of the following:

$$M_{\mu_r} = \{C, [D_{BP_0}, D_{BP_1}, ..., D_{BP_N}]\}$$

where N is the total number of agents in the pseudo-tree. This message propagates from the root to the leaves. When an agent α_κ receives this message from its parent, it removes any edge that is included in the set C. In addition, it checks the D_{BP_κ} and if is not empty, it removes the grid points that are included in D_{BP_κ} from its active domain. After the agent α_κ is done, it sends the reduction message to its children. This procedure terminates when the reduction message reaches the leaves.
The last step is to re-construct the pseudo-tree using the updated DCOP without the edges that are included in the set C. To construct the pseudo-tree, the DFS algorithm is utilized (Algorithm 1). Furthermore, the utility propagation phase and the value propagation phase of DPOP will be utilized for defining the solution.

5-6 MST-DPOP benefits

One of DPOP's main limitation is the size of the largest message, where it will be space exponential in the induced width of the pseudo-tree [30, 20]. The size of every utility message depends on the size of the separator of the corresponding agent. Hence, by reducing the size of the separator, the size of the largest message will be decreased too. As explained in this chapter, applying the MST to the DPOP can result in reducing the separator of the agents. Consequently, by extending the DPOP using the MST algorithm, the area surveillance problem can benefits the following: The MST-DPOP can bound the size of the largest message. In addition, since the size of the message is reduced, the MST-DPOP can reduce the required computation for constructing the utility message. However, due to the fact that MST-DPOP ignores some of the utility constraints, it cannot guarantee that it can define the optimal solution. Though, based on the edge-elimination technique and the MST approach, MST-DPOP can provide an error bound on the solution. That is, a bound on the difference between the solution and the global optimum. The proof of the three improvements can be found below.

5-6-1 Proof of upper bound for the message size

In order to prove that the MST algorithm provides an upper bound to the message size, the definition of the separator will be used. As previously explained, a separator of an agent consists of the parent and pseudo-parents of the current agent and the parent and pseudo-parents of its children. The MST algorithm will result in a spanning tree that does not contain any cycles. Therefore, this will guarantee that during the construction of the pseudo-tree arrangement, every agent will have at maximum one parent and no pseudo-parents. This means that the separator of every agent will be at most 1. In other words the largest message will be of dimension 1 and its size will be equal to the largest mobility domain.
Lets take for example the constraint graphs before and after the MST approach as shown in Figure 5-1a. The pseudo-tree for both constraints graphs can be seen in Figure 5-3 where the pseudo-tree on the left side represents the pseudo-tree for the constraint graph before applying the MST approach and the one on the right shows the pseudo-tree for the constraint graph after applying the MST approach.

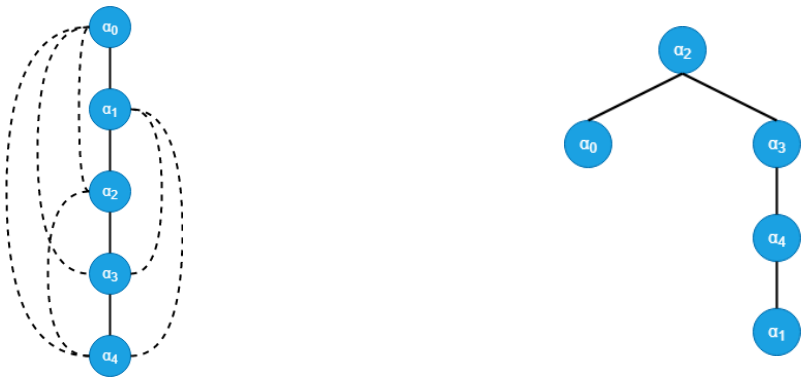


Figure 5-3: Pseudo-tree arrangement before (left) and after (right) applying the MST approach.

In this example, the separator of agent α_4 before applying the MST is $s_4 = \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\}$. Therefore, the utility message that agent α_3 has to send to agent α_2 will be of dimension 4. After applying the MST approach, the separator of agent α_4 is $s_4 = \{\alpha_3\}$ and the utility message that agent α_4 has to send to agent α_3 will be of dimension 1.

5-6-2 Proof of improvement on computational complexity

Along with the upper bound on the message size, the MST-DPOP provides improvement regarding the computational complexity. In order to derive the utility message, an agent has to consider every value assignment of the local variables of the agents in its separator. Hence, agent α_κ has to compute the utility value for v_κ combination of value assignments, where v_κ depends on the size of the domains of the agents in the separator of agent α_κ and it can be computed using Equation 4-16. The MST-DPOP reduces the size of the separator of every agent to 1. Hence, an agent has to compute the utility value for every value assignment of the local variables of its parent agent α_p. Therefore, the MST-DPOP decreases the required computations for deriving the values of the utility message. Specifically, an agent will have to compute the utility value $\eta_\kappa \times \eta_p$ times.
Lets take for example the pseudo-trees shown in Figure 5-3. Before applying the MST approach on the pseudo-tree, the agent a_4 has to compute the utility value $\eta_3 \times \eta_2 \times \eta_1 \times \eta_4$ times to construct the utility message. On the other hand, after applying the MST approach, the agent α_4 has to compute the utility value $\eta_4 \times \eta_3$ to construct the utility message. Consequently, in the MST-DPOP the agents require less computational time to compute the utility messages and the utility propagation phase can be speed up.

5-6-3 Proof of error bound on the solution

The MST-DPOP reduces the size of the largest message by eliminating edges of the constraint graph. However, this procedure affects the optimal solution for the one-step-ahead optimization. This happens because the MST-DPOP reduces the active domains of a sub-set of agents. Consequently, the optimal solution might not be available after the edge elimination procedure. For example, by removing the grid points that are included in the share domain $D_{S_{\kappa\rho}}$ from the agents α_κ and α_ρ will result in reducing the active domains of both agents. Accordingly, based on the same idea, the MST approach can provide an error bound on the solution of the MST-DPOP. More specific, it can provide an insight of the error of the solution of the MST-DPOP compared to the optimal solution.

Definition of error bound on the solution

The weight for each edge indicates the cumulative probability that the corresponding share domain has. Therefore, the error bound of the solution can be defined by using the total sum of weights of the eliminated edges. This can be done since after defining the edges that have to be removed, the edge elimination procedure blocks from the active domains of a sub-set of agents the corresponding grid points that exist in the share domain of the eliminated edges. Therefore, it could be that at the next discrete time step, the set of agents will not be able

to survey these blocked grid points.
For defining the error of the solution, first the total sum of weights T_e of the eliminated edges has to be determined. This can be done using Equation 5-6,

$$D_{EE} = \bigcup_{k \in C} D'_{S_k}$$

$$T_e = \sum_{\bar{x}_i \in D_{EE}} P(\bar{x}_i) \tag{5-6}$$

where C contains the set of the eliminated edges. Consequently, after applying the MST approach on DPOP, the solution that it will return will be at most T_e lower than the optimal solution, Equation 5-7.

$$l_b = G^* - T_e \tag{5-7}$$

where G^* is the total coverage after defining the optimal position for every mobile-sensor using DPOP. This guarantees that the solution of the MST-DPOP will be within the following range

$$l_b \leq G \leq G^* \tag{5-8}$$

with G the total coverage using MST-DPOP.
Therefore, using the total sum of the eliminated edges, an insight of the error bound of the MST-DPOP solution can be gained. More specific, using the MST-DPOP for defining the optimum position for every mobile-sensor, and by defining the total coverage, the error bound of the solution can be defined using Equation 5-9.

$$Q_G = \frac{T_e}{G + T_e} \times 100\% \tag{5-9}$$

This percentage indicates that the sub-optimum solution of the MST-DPOP is at maximum $Q_G\%$ lower than the optimal solution. In case that the T_e is equal to zero, then it implies that the MST has removed the edges with cumulative probability equals to zero, and the solution of the MST-DPOP will be the optimal solution. However, since the quality bound depends on the grid points that have been blocked from the active domains, it can get any value. If the mobile-sensors are close to each other and the area is still uncovered, the error bound will have a high estimation of error. On the contrary, when the mobile-sensors are close to each other and the area has already been covered, the error bound will have a low estimation of error. Hence, depending on the formation of the mobile-sensors, and the probability value of the surrounding grid points, the estimation of the error varies.

5-7 Conclusion

This chapter introduced an extension of DPOP where using the MST algorithm and the edge elimination procedure it can successfully decrease the size of the largest message. More specifically, it can set an upper bounds on the dimension of the utility messages since after applying the MST approach the induced width of the resulting pseudo-tree is equal to 1. Furthermore, using the edge elimination procedure, the MST-DPOP can ensure that two

mobile-sensors will not share grid points after removing the utility constraint between them. However, given the fact that the MST-DPOP reduces the active domains of the agents, the MST-DPOP cannot guarantee to find the global optimum. Based on the properties of the edge elimination procedure, an error bound of the MST-DPOP solution can be defined. The error bound can be defined using the sum of the gird points that have been blocked from the active domains of the mobile-sensors during the edge elimination procedure. Based on the sum of the blocked grid points, the error of the sub-optimum solution with respect to the optimal solution is guaranteed to be at maximum equal to this sum.
To conclude, the MST-DPOP is an extension of the DPOP solver, where it can be utilized whenever the induced width of the pseudo-tree is high. Since the MST-DPOP provides upper bounds on the utility messages, it leads to avoid exchanging exponential size of utility messages and it overcomes the high memory requirements of the DPOP. Finally, despite the fact that the MST-DPOP cannot provide the optimal solution anymore, it can define solutions with a small error compared to the optimal solution. In order to further study the performance of this extension, a comparison between the MST-DPOP and the DPOP will be conducted in the next chapter.

Chapter 6

Results

In the previous chapters, the area surveillance problem has been formulated as a Distributed Constraint Optimization Problem (DCOP). Based on the available type of solvers and the solver requirements, the Distributed Pseudo-tree Optimization Problem (DPOP) was selected to be employed for solving the DCOP. One of the DPOP disadvantages is that the largest utility message that the agents have to exchange is space exponential in the induced width of the pseudo-tree. Therefore, the higher the induced-width of the pseudo-tree is, the higher the memory requirement for the agents is. This limitation violates the algorithm requirement, that is, to be scalable with respect to the number of agents, since the agents will not be able to construct the utility messages due to the limited amount of storage that they have. To overcome this limitation of the DPOP solver, a new extension for DPOP was introduced in Chapter 5. This new solver is called MST-DPOP and it aims to reduce the size of the largest message by removing some of the utility constraints of the DCOP. Specifically, the MST-DPOP overcomes this limitation by making use of the Maximum Spanning Tree (MST) algorithm and the edge elimination procedure. The MST-DPOP is able to define the optimum solution whenever the induced width of the pseudo-tree is equal to 1. However, when the induced width of the pseudo-tree is higher than 1, the MST-DPOP cannot guarantee the optimal solution.

This chapter will present several experiments that performed to examine the performance of the MST-DPOP compared to the DPOP. First, two sets of experiments will take place. The first set of experiments will examine the quality of the solution of the MST-DPOP compared to the DPOP where the solvers will have to compute the optimum solution for the next discrete time step. The second set of experiments will analyze the number of discrete time steps that both solvers require to solve small scale area surveillance problems where only one pseudo-tree will occur. Furthermore, both solvers will be utilized for solving a larger scale area surveillance problem in order to study how both solvers would perform in a real world scenario. In the real world scenario it is possible to have several pseudo-trees at every discrete time step. Therefore, it is interesting to test how MST-DPOP will perform in such a scenario. Finally, a performance analysis regarding the scale of the problem will take place. In this analysis, by varying the number of mobile-sensors, the number of discrete time steps that both solvers require to solve the area surveillance problem will by studied.

6-1 Performance analysis of MST-DPOP

To evaluate the performance of the MST-DPOP compared to the DPOP, both solvers were utilized for solving area surveillance problems of small scale with respect to the number of mobile-sensors and the size of the area. Two series of experiments were performed, where in the first experiment the quality of the solution of both solvers was compared. In the second experiment the number of iterations that the solvers needed to solve the area surveillance problem was studied.

Quality of the solution

In the first set of experiments, small scale area surveillance problems were generated, such that they can be solved by the DPOP algorithm as well, in order to compare the DPOP solution with the solution of the MST-DPOP. The solution of the DPOP was assumed to be the global optimum in these experiments. In more detail, the mobile-sensors were placed in a rectangular area of size 40×40 grid points such that the resulting pseudo-tree has the maximum induced width. The mobile-sensors were placed such that they result in maximum induced width in order to force the MST-DPOP to remove the maximum number of utility constraints from the DCOP and analyze the quality of the solution of the MST-DPOP in a worst case scenario. Furthermore, both solvers were applied in order to define the position of every mobile-sensor for only the next discrete time step. Based on the position of every mobile-sensor, the total sum of probabilities of the grid points that the mobile-sensors had within their sensing range was computed. Finally, the percentage of the error between the DPOP and the MST-DPOP solution was determined. This error indicates the difference between the solution of the MST-DPOP and the optimum solution. In this set of experiments, along with the percentage of the error of the MST-DPOP solution, the error bound that the MST-DPOP estimates is presented.
These experiments were performed for different numbers of mobile-sensors, sensing range, and mobility range. This was done to understand the behaviour of the MST-DPOP. The results are shown in Figure 6-1.

From this set of experiments, there are three main conclusions that can be made. First, as it can be observed from the Figure 6-1, the error bound that the MST-DPOP estimates increases by increasing the mobility range. This is reasonable since by increasing the mobility range, the active domain will contain more grid points. The bigger the active domain is, the bigger the share domain will be and the more grid points will have to be blocked. Therefore, based on the definition of the error bound, the MST-DPOP will have a higher estimation of the error of the solution.
Second, the real error of the MST-DPOP solution increases by increasing the mobility range. The real error increases since by increasing the mobility range, grid points that have not been observed yet are blocked from the active domains of the mobile-sensors. Therefore, given that some unobserved gird points were blocked, the MST-DPOP was not able to define the combination of positions for the mobile-sensors such that they observe all the unobserved grid points, and the real error of the solution increases too. However, the average real error of the MST-DPOP solution in this set of experiments was within 2%. This shows that during the edge elimination procedure, the most appropriate grid points were selected to be blocked.

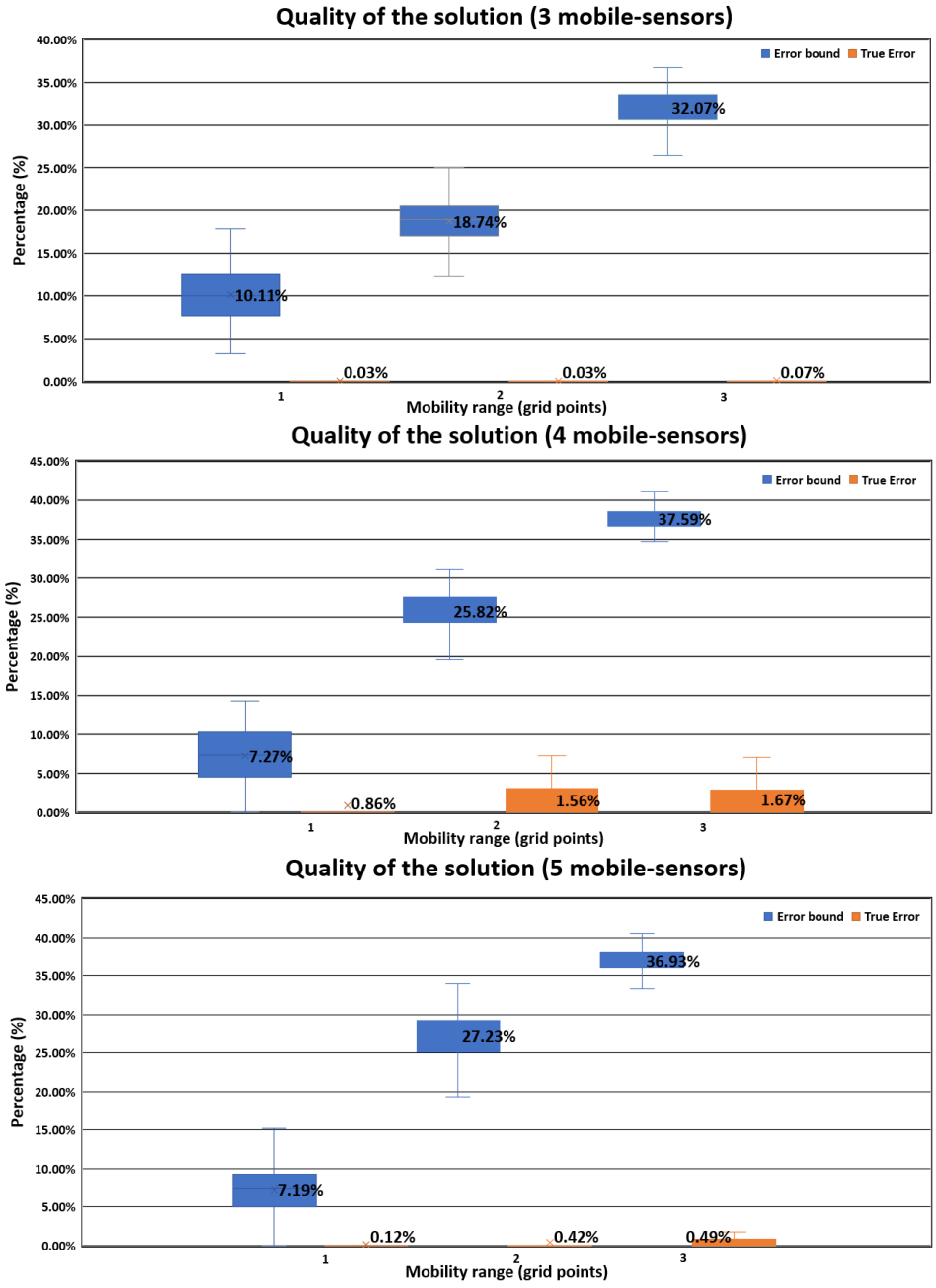


Figure 6-1: Quality experiments: Comparison between the error bound and the real error of the MST-DPOP solution compared to the DPOP solution. The error bound shows the estimate of the error of the MST-DPOP solution. The error shows the true error of the solution of the MST-DPOP. These results are the average of 100 experiments where they differ on the mobility and sensing range.

Hence, the MST-DPOP, using the reduced active domains, was still able to define the position of the mobile-sensors that resulted to solutions close to the optimum.
Finally, based on these experiments, the error bound of the MST-DPOP did not provide accurate estimates for the real error. The inaccurate estimation of the MST-DPOP was based on that the mobile-sensors were placed in the area such that they resulted in maximum induced width. Therefore, during the edge elimination procedure several unobserved grid points with probability value higher than zero had to be removed from the active domains of the mobile-sensors. Based on the definition of the error bound, this resulted in a high error estimation. However, the remaining grid points within the active domains of the mobile-sensors had higher or equal probability values than the ones that were removed, and therefore, the mobile-sensors were able to compute a sub-optimum solution with a low real error. Consequently, the error bound cannot be used for computing an accurate error bound. It can be utilized to just get an insight of what the maximum error that the solution of the MST-DPOP will have in the worst case scenario. In addition, the error bound can be utilized to define whether the solution of the MST-DPOP is the optimal one. Specifically, whenever the error bound is equal to 0%, the MST-DPOP can ensure that it will return the global optimum solution.

Number of discrete time steps

The second set of experiments was performed for comparing the number of discrete time steps that the DPOP and the MST-DPOP solvers require. Specifically, this set of experiments tested whether the lack of optimum solution of the MST-DPOP affects the required number of discrete time steps to solve the area surveillance problem. In this set of experiments, four mobile-sensors were placed in areas of different sizes. The goal of the solvers was to scan every grid point in the environment. Moreover, in this set of experiments, the sensing range and the mobility range of the mobile-sensors were selected such that at every discrete time step only one pseudo-tree exist with minimum induced width equal to 2 in order to force the MST-DPOP to remove at least 1 utility constraint from the DCOP. By doing this, the MST-DPOP will not be able to guarantee the optimal solution at every discrete time step. The results from this set of experiments can be seen in Table 6-1, where the difference in discrete time steps is the average of 100 different initial positions of the mobile-sensors.

Table 6-1: Average difference of discrete time steps that the MST-DPOP requires to solve the area surveillance compared to the DPOP solver.

Area	Sensing range	Mobility range	Difference in discrete time steps (average)	Variance
10×10	3	1	0.02	0.02
15×15	5	1	0.05	0.048
20×20	6	2	0.34	1.01
24×24	8	3	1.06	2.68
30×30	10	3	1.13	3.53

It is clear from the Table 6-1 that by increasing the mobility and sensing range, the average difference increases as well. Based on the previous experiments, the larger the mobility and sensing range is, the higher the percentage error of the MST-DPOP solution. Hence,

this implies that the MST-DPOP resulted in sub-optimum solutions at certain discrete time steps, and at the end it required more discrete time steps to solve the problem than the DPOP solver. However, for these experiments, the difference of discrete time steps on average does not surpass 2. This proves that the lack of optimum solution of the MST-DPOP can increase the required number of discrete time steps, however, the difference on average was kept low.
In the previous set of experiments, the performance of the MST-DPOP solver was compared with the DPOP solver in area surveillance problems where only one pseudo-tree existed. In real world area surveillance problems, this is not the case, since the area can be larger and therefore several constraint graphs may occur. To investigate the performance of the MST-DPOP in a real world area surveillance problem, the mobile-sensors will be employed to survey a larger area, where at every discrete time step several pseudo-trees will occur.

6-2 Area surveillance problem

To study the performance of the MST-DPOP in a real world scenario, both solvers were utilized for solving a larger scale area surveillance problem. In this experiment, six mobile-sensors were utilized for surveying a rectangular area with a size of 50×50 grid points. Their mobility range was set equal to 1 and their sensing range was equivalent to 4. The mobility and sensing range were selected such that at every discrete time step multiple constraint graphs could arise. In this case, every constraint graph represents a sub-problem that is solve independently. Moreover, it is important to mention that the MST-DPOP makes use of the MST algorithm and the edge elimination procedure whenever the induced width of the pseudo-tree is higher than one. Therefore, whenever the induced width of the pseudo-tree is equal to 1, the MST-DPOP is identical with the DPOP solver and it can define the optimum solution.
From this experiment, the percentage of the total coverage at every discrete time step was defined. The percentage of coverage defines the remaining sum of probabilities of the grid points that the mobile-sensors still had to survey. Figure 6-2 depicts the percentage of coverage of the mobile-sensors at every discrete time step.

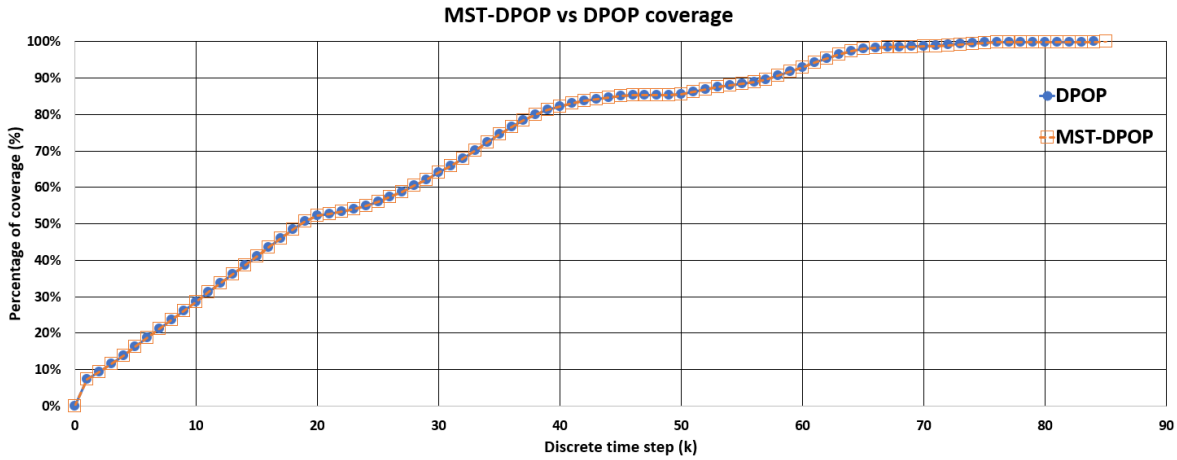


Figure 6-2: The percentage of coverage of the total area at every discrete time step. The blue colour represents the total coverage for the DPOP and the orange colour corresponds to the total coverage for the MST-DPOP.

As it can be seen from Figure 6-2, both solvers followed the same trend. However, the MST-DPOP required more discrete time steps to solve the area surveillance problem. To be more specific, the DPOP required in total 84 discrete time steps, where the MST-DPOP needed 85 discrete time steps. Based on the previous set of experiments, the average error of the solution of the MST-DPOP using a mobility range of 1 was found to be on average less than 0.12%. This explains the small difference in the number of discrete time steps that the solvers required, since even when the MST-DPOP had to remove the maximum number of utility constraints, the solution of the MST-DPOP would be 0.12% lower than the optimum one. In this experiment, the MST-DPOP was not able to define the optimum solution for the first time at the 69^{th} discrete time step. Figure 6-3 shows the percentage of coverage for both solvers after the 69^{th} discrete time step. In this figure, it is more clear that the MST-DPOP could not define the optimum solution and therefore at the end required more discrete time steps.

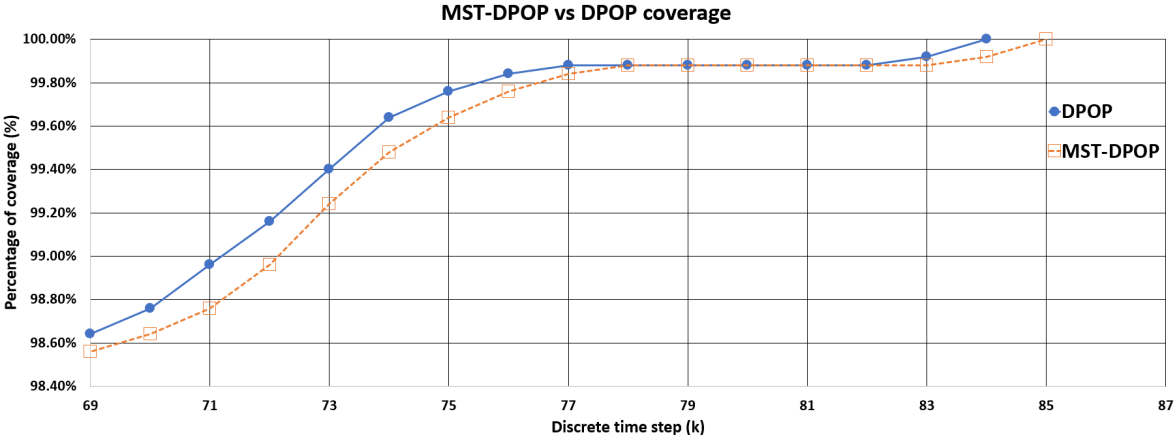


Figure 6-3: The percentage of coverage for both MST-DPOP and DPOP solver after the 69^{th} discrete time step. The blue colour represents the percentage of coverage for the DPOP and the orange colour corresponds to the percentage of coverage for the MST-DPOP.

To understand why the MST-DPOP does not return the optimum solution, the size of the messages that both solvers exchanged and the number of edges that the MST-DPOP removed from the DCOP need to be studied. Figure 6-4 shows the maximum message size that the mobile-sensors exchanged for both solvers and Figure 6-5 presents the number of edges that the MST-DPOP removed from the DCOP. It is important to mention that the computer that the experiment was performed requires 8 bytes to store a single value, plus 104 bytes for storing the values in a table. As it can be understood from Figure 6-4, after the 59^{th} discrete time step, the mobile-sensors were close to each other and they resulted in a pseudo-tree with an induced width of 3. Hence, based on Figure 6-5, the MST-DPOP had to remove more than 3 utility constraints from the DCOP. Due to the fact that until the 69^{th} discrete time step the mobile-sensors surveyed the 98% of the grid points, only a few grid points remained uncovered. Therefore, by following the edge elimination procedure for removing the grid points from the active domains of the mobile-sensors, the uncovered grid points were removed from the active domains of the mobile-sensors and the MST-DPOP solver could not define the combination of positions that result to the maximum coverage. On the contrary, in the first 7 discrete time steps, the induced width of the pseudo-tree was equal to 5. However, since only the 7% of the grid points were covered, the most of the grid points in the active domain of the

mobile-sensors were uncovered and the MST-DPOP was able to define the position for the mobile-sensors that result to the optimal solution. It is important to mention that at every discrete time step there are multiple solution that might result to the maximum coverage. Therefore, it might be that the DPOP and MST-DPOP compute a different combination of positions that result to the maximum coverage. Finally, from Figure 6-5 it is obvious that between the 12^{th} and the 36^{th} discrete time step the MST-DPOP did not have to remove any utility constraint from the DCOP, and therefore it was able to define the optimum position.

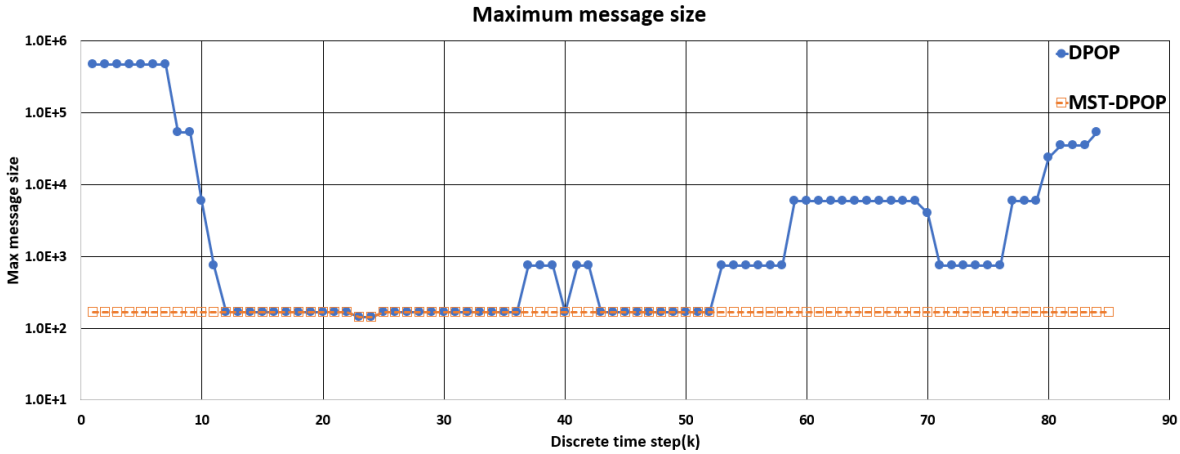


Figure 6-4: The maximum message size that the mobile-sensors exchanged at every discrete time step. The blue colour represents the maximum message size for the DPOP and the orange colour corresponds to the maximum message size for the MST-DPOP.

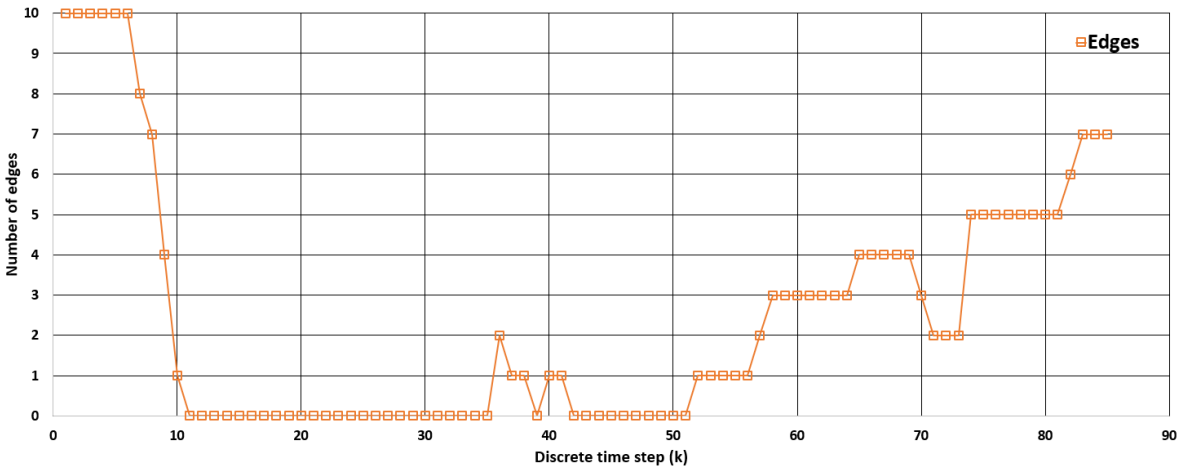


Figure 6-5: The number of edges that the MST-DPOP had to remove from the DCOP at every discrete time step so the resulting pseudo-tree had an induced-width of 1. The maximum number of edges that the MST-DPOP could remove is 10 since 6 mobile-sensors are utilized for this experiment.

For a better understanding of why the MST-DPOP returns a sub-optimum solution when the number of unobserved grid points is low, lets study the simple area surveillance example depicted in Figure 6-6. In this example, the 4 mobile-sensors are assumed to have a sensing

range equal to 1 and a mobility range equal to 2. Moreover, the green grid points are the unobserved grid points and the blue grid points are the observed ones.

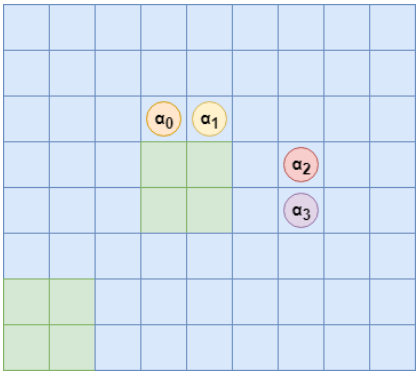
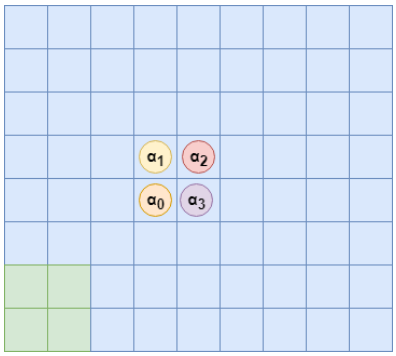
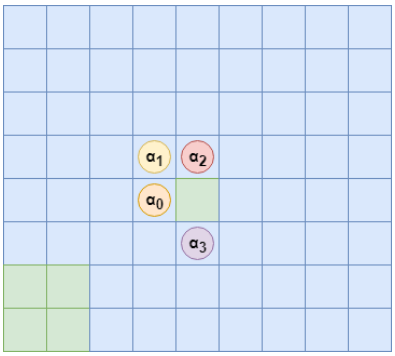


Figure 6-6: A simple area surveillance example where the dots represent the mobile-sensors, the green grid points represent the unobserved grid points and the blue grid points represent the observed ones. The mobile-sensors have a sensing range equal to 1 and a mobility range equal to 2. This will result in an instant area surveillance problem where every mobile-sensor shares a utility constraint with every other mobile-sensor.

Based on their active domains, every mobile-sensor shares a utility constraint with every other mobile-sensor. Hence, the MST-DPOP will have to remove 3 utility constraints from the DCOP. After following the MST algorithm and the edge elimination procedure, the α_3 will have to block from its active domain all of the 4 unobserved grid points. This means that the mobile-sensor α_3 will not get any utility gain if it has within its sensing range the unobserved grid points. Therefore, the 4 mobile-sensors will not be able to choose the positions in their mobility domains that result in covering all the 4 unobserved grid points and the MST-DPOP will compute a sub-optimum solution. More specifically, the solution to this problem for the next discrete time step after applying the MST-DPOP solver can be seen in Figure 6-7b. The optimal solution for this problem (using the DPOP solver) is shown in Figure 6-7a.



(a) Solution using the DPOP solver.



(b) Solution using the MST-DPOP solver.

Figure 6-7: The solution for the next discrete time step of the area surveillance problem depicted in Figure 6-6. The dots represent the mobile-sensors, the green grid points represent the unobserved grid points and the blue grid points represent the observed ones. On the left side is the solution of the problem using the DPOP solver and on the right side is the solution using the MST-DPOP.

From Figure 6-7 it is obvious that the MST-DPOP could not define the position for every mobile-sensor such that they resulted in maximum area coverage.
To conclude, even in a larger scale problem, the MST-DPOP was able to solve the area surveillance problem by using just 1 discrete time step more than the DPOP. Moreover, the MST-DPOP is more likely to result in a sub-optimum solution when the number of unobserved grid points is low and the mobile-sensors are close to each other. Finally, from this experiment, it can be seen in Figure 6-4 that the MST-DPOP managed to bound the size of the largest message and at the end it reduced the size of the largest message by several times. This indicates that the MST-DPOP solver overcomes the high memory requirements of the DPOP solver. To investigate how this improvement can affect the scalability of the MST-DPOP, a scalability analysis will take place below.

6-3 Scalability analysis

An important aspect of performance analysis of a decentralized algorithm is the study of how the algorithm's performance varies with the scale of the problem. In the area surveillance problem, the scale of the problem depends on the number of mobile-sensors. More specifically, by varying the number of agents, the DPOP and the MST-DPOP were employed to solve the area surveillance problem. In this analysis, the number of discrete time steps that every solver required to solve the problem was checked. This analysis took place in an area of a rectangular shape of dimension 50×50 grid points. Initially, all the mobile-sensors started in the middle of the area and they formed a circle. Moreover, the sensing range was set to be equal to 4, and the mobility range of the mobile-sensors was set to be 1. Finally, this analysis was performed on a computer with $8Gb$ memory. Therefore, if the message size grows larger than $8Gb$ the solver was considered unable to execute. The results from this analysis can be seen in Figure 6-8.

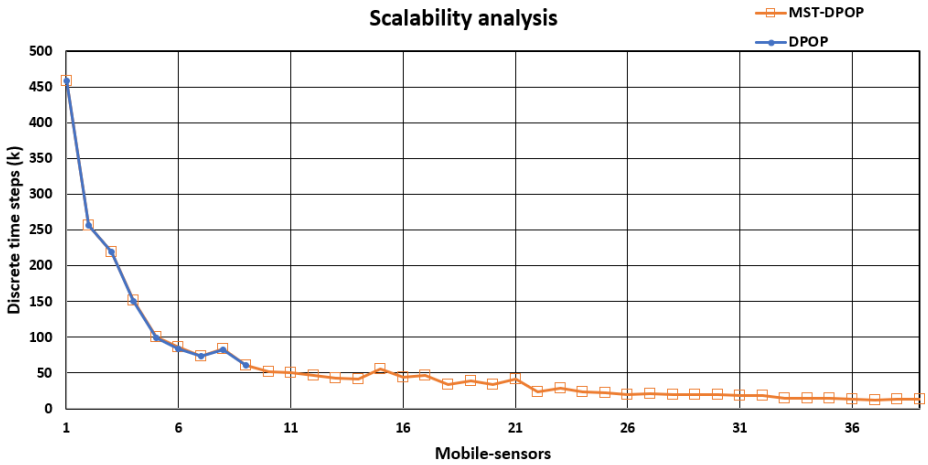


Figure 6-8: Scalability analysis for MST-DPOP and DPOP solvers. This analysis shows the number of discrete time steps that both solvers required to solve the area surveillance problem, based on the number of mobile-sensors (agents) that were utilized. The DPOP was not able to execute with more than 9 mobile-sensors due to the fact that the size of the largest message was larger than the available memory.

From the scalability analysis, it is noticeable that by increasing the number of mobile-sensors, the MST-DPOP and the DPOP solvers required less number of discrete time steps. Furthermore, as it can be seen in Figure 6-8, the DPOP solver was not able to solve the area surveillance problem with more than 9 mobile-sensors. This is due to the fact that the largest utility message that the agents had to construct was larger than $8Gb$ and therefore the computer could not construct the utility message. This makes the DPOP unable to execute the utility propagation procedure and at the end the DPOP failed to solve the area surveillance problem. On the other hand, the MST-DPOP was able to solve the problem with more than 30 mobile-sensors. The MST-DPOP bounds the size of the largest message and it does not require large amount of memory to construct the utility messages. Consequently, the scalability analysis proves that the MST-DPOP does not have any restriction on the number of mobile-sensors that can be used since its largest utility message does not depend on the number of the mobile-sensors.

6-4 Discussion

In this chapter, several experiments were performed to study the performance of the MST-DPOP solver compared to the DPOP solver. Based on the experiments, the solution of the MST-DPOP solver, whenever the induced width of the pseudo-tree was higher than 1, was found to have on average an error of less than 2% from the optimum solution. In addition, it was found that despite the fact that the MST-DPOP solver cannot guarantee the optimum solution, it requires on average 1 discrete time step more than the DPOP.
Furthermore, based on the real world area surveillance problem, the MST-DPOP was found to perform as good as the DPOP when the number of unobserved grid points was higher than the observed ones. This happened due to the fact that the remaining grid points in the active domains of the mobile-sensors after following the edge elimination procedure were still unobserved, and therefore, the MST-DPOP could compute to position of the mobile-sensors that result to the maximum coverage. However, when the number of the unobserved grid points was low and the MST-DPOP had to remove utility constraints from the problem, it was more likely for the MST-DPOP solver to compute a sub-optimum solution. This mainly happened since the MST-DPOP blocked from the active domains of the mobile-sensors the remaining unobserved grid points and it was not able to define the combination of positions that resulted in maximum coverage. This leads to the conclusion that during the first discrete time steps, where every grid point has not been observed, the MST-DPOP should be preferred over the DPOP in order to avoid exchanging exponential size of utility messages. On the other hand, when the number of unobserved grid points is less than 10% of the total number of grid points, the DPOP solver should be adopted to avoid increasing the required discrete time steps for solving the area surveillance problem.
Finally, based on the scalability analysis, the improvement on the memory requirements of the MST-DPOP solver makes the MST-DPOP solver scalable with respect to the number of mobile-sensors that can be used for solving the area surveillance problem.
To conclude, based on the experiments that took place in this chapter and given the improvements that the MST-DPOP offers over the DPOP, the MST-DPOP solver can be considered as more appropriate for solving the area surveillance problem than the DPOP solver. Moreover, the MST-DPOP solver was found to satisfy all of the solver's requirement, since even when it was not able to define the optimal solution, the average error of its solution was less

than 5%. However, the more ideal scenario for solving the area surveillance problem in an optimal manner would be to use the MST-DPOP solver as soon as the percentage of coverage is less than 90%, and then to employ the DPOP solver for surveying the final 10% of the area.

Chapter 7

Conclusions and future work

In this thesis, the problem of the area surveillance was addressed. The area surveillance is assumed to take place in an unobstructed area which is discretized into multiple grid points. Moreover, mobile-sensors were utilized for surveying the area. Mobile-sensors are robots that are equipped with onboard computers and cameras, so they can perceive their surrounding environment. In order to solve the area surveillance problem, several mobile-sensors have to coordinate their actions and survey every unobserved grid point of the area under surveillance. This problem is a constraint optimization problem, where its solution is a waypoint for every mobile-sensor. Moreover, due to the restricted computation resources of the onboard computers, and due to the fact that the algorithm should be scalable with respect to the number of mobile-sensors, a centralized approach could not be used. For these reasons, in the thesis, a decentralized algorithm was designed such that it can run online and on the onboard computers, and it can compute waypoints for every mobile-sensor until they manage to survey every unobserved grid point of the environment.

7-1 Conclusions

As a first step, a coordination model was needed. The coordination model determines the relations between the different mobile-sensors and it defines whether two mobile-sensors will have to communicate for computing their waypoints for the next discrete time step. One of the most important formalism for coordination that can be found in the literature is the Distributed Constraint Optimization Problem (DCOP). The DCOP has the ability to define the problem by utilizing the constraints between the agents. The solution of a DCOP is a value assignment for every variable in the problem such that the aggregate utility of the utility constraints is maximized. Based on the available solvers of DCOPs, and given the fact that DCOP can be employed for multi-agent systems, the area surveillance problem was formulated as DCOP.
The next step was to choose a solver that best fits the requirements for the area surveillance

problem. In more details, the solver should be able to solve the problem in a decentralized manner. Moreover, the solver should be able to result at every discrete time step in solutions that their error is less than 5% from the optimum solution. Finally, the solver should be scalable with respect to the number of mobile-sensors. Based on the available type of solvers, the complete inference based was found to best fit the requirements.
One of the most well known complete inference based solvers is the Distributed Pseudo-tree Optimization Procedure (DPOP) [20]. The DPOP is able to solve a DCOP in a decentralized manner using a linear number of massages with respect to the number of agents. However, its main disadvantage lies on the memory requirements of the agents. More precisely, during the utility propagation phase, the largest message that the agents have to exchange is space exponential in the induced width of the pseudo-tree. Consequently, if the induced width of the pseudo-tree is high, the agents require large amount of memory for constructing and sending the utility messages to the agents that share a utility constraint. Given the fact that the agents in the area surveillance problem are mobile-sensors with a limited available of storage, they might be unable to construct the utility messages. This means that the agents will be unable to exchange utility messages and the DPOP solver will fail to solve the area surveillance problem. Consequently, this limit the number of mobile-sensors that can be utilized for solving the area surveillance problem. To overcome this limitation, this thesis introduced an extension for the DPOP. The new solver is called MST-DPOP.

The MST-DPOP

The MST-DPOP intends to bound the size of the largest message by removing the least important utility constraints from the DCOP. To do so, the MST-DPOP makes use of the Maximum Spanning Tree (MST) algorithm, and a novel procedure called edge elimination procedure. First, the MST algorithm defines the utility constraints from the constraint graph that need to be removed such that the resulting pseudo-tree will have an induced width of 1. In addition, the edge elimination procedure, using the utility constraints that the MST discarded from the DCOP, blocks from the active domain of the mobile-sensors the corresponding grid points that belong to the utility constraints. By removing these grid points from the active domain of the mobile-sensors, the MST-DPOP ensures that the mobile-sensors will not choose the potential positions that drive them towards the same region of the area. The MST-DPOP overcomes the memory requirement of the DPOP solver. However, the MST algorithm along with the edge elimination procedure makes the MST-DPOP an incomplete solver due to the fact that whenever the induced width of the pseudo-tree is larger than 1 the MST-DPOP ignores some utility constraints and at the end it cannot guarantee the optimal solution.
To investigate the performance of the MST-DPOP and the DPOP solver, several experiments were performed. The first set of experiments tried to examine the quality of the solution of the MST-DPOP compared to the optimal solution. The optimal solution is the solution of the DPOP solver. In this set of experiments, both solvers had to define the position of the mobile-sensors that maximizes the total coverage based on the constraints. Moreover, during this set of experiments the mobility and the sensing range was varied to check whether the quality of the solution is affected. It was found that by increasing the mobility and the sensing range, the percentage of the error was increasing too. This is due to the fact that by increasing these ranges, the active domain increases too. Therefore, during the edge elimination procedure

more grid points had to be removed from the active domains of the mobile-sensors and the error from the optimum position was higher. Nonetheless, from this set of experiments, it was found that the MST-DPOP was able to define a solution with an average error of less than 2%. Consequently, the MST-DPOP solver, using the MST algorithm and the edge elimination procedure, was able to define the most appropriate grid points to remove from the active domains of the mobile-sensors such that it reduced the size of the largest message and at the same time it kept the error of the solution low. This set of experiments proved also that the MST-DPOP satisfied the solver's requirement that the solution should be within 5% from the optimal solution.

Furthermore, the next set of experiments aimed to prove that due to the low error of the solution of the MST-DPOP solver, the number of discrete time steps that the MST-DPOP will require to solve the area surveillance problem will be very close to that of the DPOP solver. After running several set of experiments, and by comparing the number of discrete time steps that both solvers require, it was found that the MST-DPOP required on average 1 discrete time step more than the DPOP solver to observe every unobserved grid point. Consequently, this set of experiments proved that by keeping the error of the solution low, the required number of discrete time steps can be kept low as well.

In addition, the performance of the MST-DPOP solver against the DPOP solver was analyzed in a larger scale area surveillance problem. This was done to investigate the performance of MST-DPOP in a real world scenario. In the case of a real world scenario, it is possible that multiple constraints graph could arise where every constraint graph should solve the problem independently. Based on the results from this experiment, whenever the MST-DPOP had to remove utility constraints from the DCOP and the number of unobserved grid points was still high, the MST-DPOP was able to define the optimal solution. This mainly happened since even by removing several grid points from the active domain of the mobile-sensors, the remaining grid points were still unobserved and the solver was able to define a position for every mobile-sensors such that they achieved maximum coverage. On the contrary, whenever the MST-DPOP had to remove utility constraints from the DCOP and the number of unobserved grid points was low, the MST-DPOP could not define the optimal position. This happened because during the edge elimination procedure, the uncovered grid points were removed from the active domains of the mobile-sensors, and the MST-DPOP was not able to define a combination of positions such that the mobile sensors achieved maximum coverage.

Finally, based on the scalability analysis, it was proved that the new extension was able to overcome the exponential memory requirements that the DPOP has. The MST-DPOP was able to solve the area surveillance problem using more than 30 mobile-sensors compared to the DPOP that could not solve the problem with more than 9 due to the limited available memory of the computer ($8Gb$). Hence, based on this set of experiments, the MST-DPOP solver satisfies the scalability requirement of the MST-DPOP.

To conclude, the MST-DPOP satisfies all of the three requirements of the solver since in the worst case scenario, the MST-DPOP was able to compute a solution with an error less than 5%. Based on the set of experiments that performed in the last chapter, the most ideal scenario for solving the area surveillance problem in an optimal manner, it is to employ the MST-DPOP initially, where the unobserved grid points are more than the observed grid points, and then to employ the DPOP solver for surveying the last part of the area under surveillance. This will lead to avoid exchanging exponential size of messages in the initial stage, where the mobile-sensors start in the middle of the area under surveillance, and it will

lead to minimize the number of required discrete time steps at the end, where the MST-DPOP was not able to define the optimal solution. However, in the case that the induced width of the pseudo-tree is high and will result in messages higher than the available storage of the mobile-sensors, the MST-DPOP should be employed.

7-2 Future work

Based on the conclusions and the fact that a one-step-ahead optimization was used to solve the problem, it can be concluded that there is still room of improvement for this solver.
First, in the MST-DPOP, for defining the utility constraints that have to be removed from the DCOP, the MST algorithm was employed. The MST algorithm is able to define the spanning tree that results to the maximum sum of weights and it has the lowest induced width. The performance of the MST-DPOP can be improved by employing a different approach for defining the utility constraints that have to be removed. More specific, instead of using the MST algorithm, a new algorithm can be designed which will be able to define the minimum number of utility constraints that can be removed from the DCOP such that the mobile-sensors will not run out of memory during the construction of the utility messages. This is challenging, since it is not directly clear how a utility constraint can reduce the induced width of the resulting pseudo-tree. Moreover, it is also challenging to define what is the minimum number of utility constraints that have to be removed from the DCOP such that the mobile-sensors will be able to construct the utility messages.
Furthermore, using a one-step-ahead optimization allows the mobile-sensors to run the required computations on their onboard computers. However, during the optimization, the mobile-sensors consider the impact that only their next action will have to the problem. By increasing the optimization horizon can decrease the required number of discrete time steps to solve the problem. This will increase the required computational power, where it will also increase the utility constraints between the mobile-sensors.

Bibliography

[1] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "Adopt: asynchronous distributed constraint optimization with quality guarantees," *Artificial Intelligence*, vol. 161, no. 1, pp. 149 – 180, 2005.

[2] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings, "Bounded approximate decentralised coordination via the max-sum algorithm," *Artificial Intelligence*, vol. 175, no. 2, pp. 730–759, 2011.

[3] M. Vinyals, J. A. Rodriguez-Aguilar, and J. Cerquides, "A survey on sensor networks from a multiagent perspective," *The Computer Journal*, vol. 54, no. 3, pp. 455–470, 2011.

[4] S. Poduri and G. S. Sukhatme, "Constrained coverage for mobile sensor networks," in *IEEE International Conference on Robotics and Automation. Proceedings.*, vol. 1, pp. 165–171, IEEE, 2004.

[5] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings, "Decentralised coordination of mobile sensors using the max-sum algorithm.," in *IJCAI*, vol. 9, pp. 299–304, 2009.

[6] S. H. Semnani and O. A. Basir, "Multi-target engagement in complex mobile surveillance sensor networks," *Unmanned Systems*, vol. 5, no. 01, pp. 31–43, 2017.

[7] R. Zivan, R. Glinton, and K. Sycara, "Distributed constraint optimization for large teams of mobile sensing agents," in *Web Intelligence and Intelligent Agent Technologies. IEEE/WIC/ACM International Joint Conferences*, vol. 2, pp. 347–354, 2009.

[8] C. J. Carpenter, R. N. Lass, E. Sultanik, C. J. Dugan, G. Naik, P. J. Modi, J. B. Kopena, D. N. Nguyen, and W. C. Regli, "Disaster evacuation support," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, p. 263, ACM, 2007.

[9] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *International journal of computer vision*, vol. 38, no. 1, pp. 15–33, 2000.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[11] A. R. Leite, F. Enembreck, and J.-P. A. Barthes, "Distributed constraint optimization problems: Review and perspectives," *Expert Systems with Applications*, vol. 41, no. 11, pp. 5139–5157, 2014.

[12] N. R. Jennings and S. Bussmann, "Agent-based control systems: Why are they suited to engineering complex systems?," *IEEE control systems*, vol. 23, no. 3, pp. 61–73, 2003.

[13] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[14] P. Scerri, R. Vincent, and R. Mailler, *Coordination of large-scale multiagent systems*. Springer, 2006.

[15] W. Yeoh and M. Yokoo, "Distributed problem solving," *AI Magazine*, vol. 33, no. 3, p. 53, 2012.

[16] A. Farinelli, A. Rogers, and N. R. Jennings, "Agent-based decentralised coordination for sensor networks using the max-sum algorithm," *Autonomous agents and multi-agent systems*, vol. 28, no. 3, pp. 337–380, 2014.

[17] S. H. Semnani and O. A. Basir, "Target to sensor allocation: A hierarchical dynamic distributed constraint optimization approach," *Computer Communications*, vol. 36, no. 9, pp. 1024–1038, 2013.

[18] R. Junges and A. L. Bazzan, "Evaluating the performance of dcop algorithms in a real world, dynamic problem," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pp. 599–606, 2008.

[19] F. Enembreck and J.-P. A. Barthès, "Distributed constraint optimization with mulbs: A case study on collaborative meeting scheduling," *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 164–175, 2012.

[20] A. Petcu and B. Faltings, "A scalable method for multiagent constraint optimization," tech. rep., 2005.

[21] R. T. Maheswaran, J. P. Pearce, and M. Tambe, "Distributed algorithms for dcop: A graphical-game-based approach," in *ISCA PDCS*, 2004.

[22] T. Le, T. C. Son, E. Pontelli, and W. Yeoh, "Solving distributed constraint optimization problems using logic programming," *Theory and Practice of Logic Programming*, vol. 17, no. 4, pp. 634–683, 2017.

[23] V. Kumar, "Algorithms for constraint-satisfaction problems: A survey," *AI magazine*, vol. 13, no. 1, pp. 32–32, 1992.

[24] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 498–519, 2001.

[25] R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, and K. Sycara, "Distributed constraint optimization for teams of mobile sensing agents," *Autonomous Agents and Multi-Agent Systems*, vol. 29, pp. 495–536, May 2015.

[26] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings, "Decentralised coordination of low-power embedded devices using the max-sum algorithm," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pp. 639–646, 2008.

[27] W. Yeoh, P. Varakantham, X. Sun, and S. Koenig, "Incremental dcop search algorithms for solving dynamic dcop problems," in *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, vol. 2, pp. 257–264, 2015.

[28] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[29] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems*, pp. 5330–5340, 2017.

[30] A. Petcu, *A class of algorithms for distributed constraint optimization*, vol. 194. Ios Press, 2009.

[31] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[32] W. Yeoh, A. Felner, and S. Koenig, "Bnb-adopt: An asynchronous branch-and-bound dcop algorithm," *Journal of Artificial Intelligence Research*, vol. 38, pp. 85–133, 2010.

[33] Z. Chen, Z. He, and C. He, "An improved dpop algorithm based on breadth first search pseudo-tree for distributed constraint optimization," *Applied Intelligence*, vol. 47, no. 3, pp. 607–623, 2017.

[34] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg, "Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks," *Artificial Intelligence*, vol. 161, no. 1, pp. 55 – 87, 2005.

[35] F. Fioretto, T. Le, W. Yeoh, E. Pontelli, and T. C. Son, "Improving dpop with branch consistency for solving distributed constraint optimization problems," in *International Conference on Principles and Practice of Constraint Programming*, pp. 307–323, Springer, 2014.

[36] A. Kumar, A. Petcu, and B. Faltings, "H-dpop: Using hard constraints for search space pruning in dcop.," in *AAAI*, pp. 325–330, 2008.

[37] A. Petcu and B. Faltings, "Mb-dpop: A new memory-bounded algorithm for distributed optimization.," in *IJCAI*, pp. 1452–1457, 2007.

[38] R. J. Bayardo and D. P. Miranker, "On the space-time trade-off in solving constraint satisfaction problems," in *International joint conference on artificial intelligence*, vol. 14, pp. 558–562, 1995.

[39] I. Brito and P. Meseguer, "Improving dpop with function filtering," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 141–148, 2010.

[40] A. Petcu and B. Faltings, "Odpop: An algorithm for open/distributed constraint optimization," in *AAAI*, vol. 6, pp. 703–708, 2006.

[41] A. Petcu and B. Faltings, "Approximations in distributed optimization," in *International Conference on Principles and Practice of Constraint Programming*, pp. 802–806, Springer, 2005.

[42] M. Vinyals, J. A. Rodriguez-Aguilar, and J. Cerquides, "Generalizing dpop: Action-gdl, a new complete algorithm for dcops," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 1239–1240, 2009.

[43] F. V. Jensen and F. Jensen, "Optimal junction trees," in *Uncertainty Proceedings*, pp. 360–366, Elsevier, 1994.

[44] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Transactions on Programming Languages and systems*, vol. 5, no. 1, pp. 66–77, 1983.

[45] S. Rai and S. Sharma, "Determining minimum spanning tree in an undirected weighted graph," in *International Conference on Advances in Computer Engineering and Applications*, pp. 637–642, IEEE, 2015.

[46] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[47] R. C. Prim, "Shortest connection networks and some generalizations," *Bell system technical journal*, vol. 36, no. 6, pp. 1389–1401, 1957.