**UNIVERSITY OF SOUTHAMPTON**
Faculty of Engineering and Applied Science
School of Electronics and Computer Science

# Decentralised Coordination of Information Gathering Agents

by Ruben Stranders

Supervisors: Prof. Nicholas R. Jennings and Dr. Alex Rogers
Examiners: Prof. Victor R. Lesser and Dr. Seth Bullock

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

November 2010

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Ruben Stranders

Unmanned sensors are rapidly becoming the *de facto* means of achieving situational awareness—the ability to make sense of, and predict what is happening in an environment—in disaster management, military reconnaissance, space exploration, and climate research. In these domains, and many others besides, their use reduces the need for exposing humans to hostile, impassable or polluted environments. Whilst these sensors are currently often pre-programmed or remotely controlled by human operators, there is a clear trend toward making these sensors fully autonomous, thus enabling them to make decisions without human intervention.

Full autonomy has two clear benefits over pre-programming and human remote control. First, in contrast to sensors with pre-programmed motion paths, autonomous sensors are better able to adapt to their environment, and react to *a priori* unknown external events or hardware failure. Second, autonomous sensors can operate in large teams that would otherwise be too complex to control by human operators. The key benefit of this is that a team of cheap, small sensors can achieve through cooperation the same results as individual large, expensive sensors—with more flexibility and robustness.

In light of the importance of autonomy and cooperation, we adopt an agent-based perspective on the operation of the sensors. Within this view, each sensor becomes an *information gathering agent*. As a team, these agents can then direct their collective activity towards collecting information from their environment with the aim of providing accurate and up-to-date situational awareness.

Against this background, the central problem we address in this thesis is that of achieving accurate situational awareness through the coordination of multiple information gathering agents. To achieve general and principled solutions to this problem, we formulate a generic problem definition, which captures the essential properties of dynamic environments. Specific instantiations of this generic problem span a broad spectrum of concrete application domains, of which we study three canonical examples: monitoring environmental phenomena, wide area surveillance, and search and patrol.

The main contributions of this thesis are decentralised coordination algorithms that solve this general problem with additional constraints and requirements, and can be grouped into two categories. The first category pertains to decentralised coordination of *fixed* information gathering agents. For these agents, we study the application of decentralised coordination during two distinct phases of the agents' life cycle: *deployment* and *operation*. For the former, we develop an efficient algorithm for maximising the quality of situational awareness, while simultaneously constructing a reliable communication network between the agents. Specifically, we present a novel approach to the NP-hard problem of frequency allocation, which deactivates certain agents such that the problem can be provably solved in polynomial time. For the latter, we address the challenge of coordinating these agents under the additional assumption that their control parameters are continuous. In so doing, we develop two extensions to the max-sum message passing algorithm for decentralised welfare maximisation, which constitute the first two algorithms for distributed constraint optimisation problems (DCOPs) with continuous variables—CPLF-MS (for linear utility functions) and HCMS (for non-linear utility functions).

The second category relates to decentralised coordination of *mobile* information gathering agents whose motion is constrained by their environment. For these agents, we develop algorithms with a *receding* planning horizon, and a *non-myopic* planning horizon. The former is based on the max-sum algorithm, thus ensuring an efficient and scalable solution, and constitutes the first online agent-based algorithm for the domains of pursuit-evasion, patrolling and monitoring environmental phenomena. The second uses sequential decision making techniques for the offline computation of *patrols*—infinitely long paths designed to continuously monitor a dynamic environment—which are subsequently improved on at runtime through decentralised coordination.

For both topics, the algorithms are designed to satisfy our design requirements of quality of situational awareness, adaptiveness (the ability to respond to *a priori* unknown events), robustness (the ability to degrade gracefully), autonomy (the ability of agents to make decisions without the intervention of a centralised controller), modularity (the ability to support heterogeneous agents) and performance guarantees (the ability to give a lower bound on the quality of the achieved situational awareness). When taken together, the contributions presented in this thesis represent an advance in the state of the art of decentralised coordination of information gathering agents, and a step towards achieving autonomous control of unmanned sensors.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Nomenclature

| | |
|---|---|
| $\mathbf{A}$ | A collection of agents |
| $\mathcal{A}_i$ | A single agent |
| $\mathcal{G} = (V, E)$ | A graph encoding the physical layout of an environment, where $E$ encodes the possible movements between locations $V$ |
| $adj_{\mathcal{G}}(v)$ | The set of vertices adjacent to $v$ in a graph $\mathcal{G}$ |
| $\mathrm{d}(u, v)$ | The Euclidian distance between coordinates $u$ and $v$ |
| $\mathrm{d}_G(u, v)$ | The shortest path (Dijkstra) distance between vertices $u$ and $v$ in graph $G$ |
| $\mathrm{diam}(G)$ | The diameter of a graph $G$ |
| $U_i$ | The utility function of agent $\mathcal{A}_i$ |
| $p_i$ | The control or action variable of agent $\mathcal{A}_i$ |
| $\mathbf{p}_i$ | The set of control variables that influence the utility agent $\mathcal{A}_i$, or, equivalently, the set of parameters to utility function $U_i$ |
| $\mathcal{D}_i$ | The domain of action variable $p_i$ of agent $\mathcal{A}_i$ |
| $a_i^j$ | The $j^{\text{th}}$ action in domain $\mathcal{D}_i$ |
| $q_{i \to j}(p_i)$ | A max-sum message sent by variable $p_i$ to function $U_j$ |
| $r_{j \to i}(p_i)$ | A max-sum message sent by function $U_j$ to variable $p_i$ |
| $\bigcirc_i$ | The observation area of agent $\mathcal{A}_i$ |
| $O_i^t$ | The observations made by agent $i$ at time $t$ |
| $O_{\mathbf{A}}^t$ | The observations made by all agents at time $t$ |
| $\mathbf{O}_{\mathbf{A}}^t$ | The observations made by all agents at or before time $t$ |
| $o(l)$ | The spatial coordinate of observation $o$ |

| | |
|---|---|
| $o(t)$ | The temporal coordinate of observation $o$ |
| $o(m)$ | The actual measurement associated $o$ |
| $\mathbf{C}$ | A set of graph clusters |
| $\mathbf{T}$ | A set of transfer nodes |
| $\lambda_C$ | The number of time steps since cluster $C$ was last visited |
| $B$ | The maximum number of time steps allocated to an agent to patrol a cluster |
| $\rho_A(B)$ | The incremental value of observation set $A$ when added to $B$. Formally, $\rho_A(B) = f(A \cup B) - f(B)$ |

# Declaration of Authorship

I, Ruben Stranders, declare that the thesis entitled *Decentralised Coordination of Information Gathering Agents* and the work presented in this thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published in a number of conference and journal papers (see Section 1.2 for a list).

**Signed:**

**Date:**

# Acknowledgements

I would like to thank all of the people who encouraged and supported me during the undertaking of my research.

First and foremost, I would like to thank my supervisors Nick Jennings and Alex Rogers to whom I am greatly indebted for their unwavering support, guidance, encouragement and expertise. I am incredibly thankful the opportunity they have provided me to do this research at the School of Electronics and Computer Science at University of Southampton, and for gently nudging me towards this fascinating field of research that lies at the nexus of agent-based systems and autonomous sensors.

I would also like to thank my co-authors with whom I have had the distinct pleasure of working: Alessandro Farinelli for introducing me to the max-sum algorithm, Francesco Delle Fave for suggesting the application of my work to the domains of pursuit-evasion and patrolling, Tom Voice for collaborating on the Hybrid Continuous Max-Sum algorithm, and Enrique Munoz de Cote for his expertise on sequential decision making. Their advice and fruitful discussions have proved invaluable and shaped my research to a great extent.

Thanks goes to my (former) colleagues at the Intelligence, Agents, Multimedia group for their support and friendship and for providing the necessary distractions: Alessandro Farinelli, Archie Chapman, Enrico Gerding, Enrique Munoz de Cote, Francesco Delle Fave, George Chalkiadakis, Gopal Ramchurn, Luke Teacy, Maria Polukarov, Muddasser Alam, Perukrishnen Vytelingum, Ramachandra Kota, Sebastian Stein, Simon Williamson, Talal Rahwan, Tom Voice and Zinovi Rabinovich. I owe special thanks to Rocío Aldeco Perez, who at the time of this writing is doing her last thesis corrections. Her unique perspective and support have been invaluable during the course of writing this thesis, and I cherish her friendship.

I gratefully acknowledge the financial support by BAE Systems and EPSRC through the ALADDIN project.

My greatest debt is to my parents and brother, who have patiently supported throughout this process. This thesis would never have been written without their encouragement and moral support, and I am deeply grateful to have had their guidance throughout my life.

# Chapter 1

# Introduction

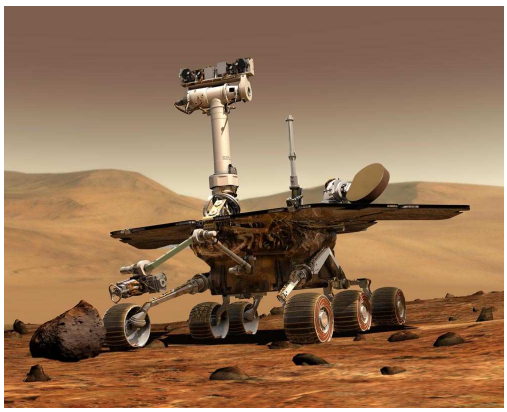
Recent years have seen an explosive growth in the use of both fixed and mobile sensors in a broad spectrum of application domains, ranging from oceanography, climate research, and space exploration, to security, disaster management and military operations. Within these domains, sensors are brought into action to reduce the need for human presence in remote or hostile environments, and to enable monitoring over extended periods of time. Working together as a team, rather than as a collection of individuals, these sensors are able to share information and coordinate their actions in order to increase the accuracy and resolution of the picture that they compile of their environment.

The varied nature of these application domains is reflected by the diversity of the sensors themselves. This not only comprises the different physical quantities they measure, and the way in which they are powered, but also their mobility. Some examples include fixed sensors powered by solar radiation for measuring flood-levels (Figure 1.1(a)), autonomous ground vehicles (AGVs) for space exploration (Figure 1.1(b)), autonomous aerial vehicles (UAVs) for military surveillance (Figure 1.1(c)), and autonomous underwater vehicles (UUVs) for mine detection (Figure 1.1(d)). Clearly, these sensors operate in different domains, and are used to accomplish different missions within them, such as:

- Monitoring dynamic and uncertain environmental conditions, such as radiation, temperature and gas concentrations.
- Finding a moving target, be it cooperative (i.e. a wounded civilian in a disaster scenario), or uncooperative (i.e. an attacker in a military scenario).
- Patrolling a building or perimeter to prevent intrusions.

Abstracting from the specifics of these different missions, we can consider each of these as an endeavour to provide *situational awareness* (Endsley 1995). Roughly speaking,

(a) A Floodnet Sensor[1]



(b) The Mars Rover[2]

(c) The Predator UAV[3]


(d) The Talisman UUV[4]

FIGURE 1.1: Some examples of fixed and mobile sensors.

situational awareness involves the understanding of events and its impact on the objectives that exist within the scenario, both now and in the future. As such, it is of critical importance in sensitive scenarios, such as disaster response, where a misunderstanding of the state of the environment could lead to injury or the loss of life, but also in scenarios that are less time constrained, but where a deep understanding of the phenomena that exist within an environment can lead to novel and important insights (e.g. climate research, oceanography, and space exploration).

Given the potentially high stakes and sensitivity of these application domains, it is important that these sensors execute their missions in a timely fashion, while being robust against failures and able to adapt to unforeseen events. Thus, it is desirable that these sensors operate *autonomously*—taking their own decisions based on information that is available locally, or through communication with other sensors—for two reasons. First, the absence of autonomy implies the existence of a central controller, which constitutes a single point of failure, or reliance on a human operator, whose attention is a scarce

[1]Source: `http://envisense.org/floodnet/pictures/node2hightide.htm`
[2]Source: `http://www.jpl.nasa.gov/news/features.cfm?feature=958`
[3]Source: `http://www.af.mil/shared/media/photodb/photos/081131-F-7734Q-001.jpg`
[4]Source: `http://www.baesystems.com/ProductsServices/autoGen_106919171313.html`

resource that could, in most cases, be better utilised in pursuit of the mission goals. Second, an autonomous sensor is capable of reacting quickly to changes in its environment, since it does not need to communicate all data necessary to compute its control inputs to this central controller—a process that costs valuable time and consumes scarce communication bandwidth and power. Indeed, these considerations have led to the United States Air Force to call for fully autonomous UAVs by 2047 in their long-term strategy on autonomous vehicles (United States Air Force 2009).

In light of the importance of autonomy, the use of *agent-based technologies* has been advocated to control sensors in a fully decentralised fashion (Rogers et al. 2009). This means that control is distributed over multiple computational entities, and thus no centralised controller exists. Within the agent-based perspective, each sensor becomes an *intelligent agent*: "a computer system that is capable of *flexible* autonomous action in order to meet its design objectives", which "should perceive [its] environment and respond in a timely fashion to changes that occur in it", exhibits "opportunistic goal-directed behaviour", and "should be able to interact [...] with other artificial agents and humans in order to complete their own problem solving and to help others with their activities" (Wooldridge & Jennings 1995). In this thesis, we shall refer to these sensing agents as *information gathering agents*:

**Definition 1.1** (Information Gathering Agent)**.** An information gathering agent is an intelligent agent that directs its activity towards collecting information from its environment with the aim of providing high quality and up-to-date situational awareness.

Operating as a team, agents embedded in small, cheap sensors can collectively achieve through coordination the same results as agents embedded in individual large, expensive sensors—with more flexibility and robustness. This is clearly a desirable trait in the aforementioned scenarios. Such a *team* of information gathering agents can be regarded as a *cooperative multi-agent system*, in which multiple agents coordinate in a decentralised fashion in order to collectively achieve their aims. Concretely, coordination between mobile agents entails choosing their trajectories through their environment (e.g. airspace, road networks, or buildings) in order to maximise their information gain, by minimising redundant sensing coverage of the area. Similarly, for fixed agents that cannot reposition themselves, coordination is required to maximise coverage by scheduling their activation schedules in time, or by adjusting the viewing direction of their sensors.

Coordinating a team of agents in the aforementioned hostile environments is a difficult challenge. Care must be taken to develop algorithms that are robust to the loss of communication and that degrade gracefully in the event of the failure of one or more agents, while at the same time providing accurate situational awareness. It is this specific challenge that we address in this thesis, which, in its simplest form, can be formulated as follows:

*How to coordinate a team of information gathering agents in a decentralised fashion, so as to maximise the quality of the situational awareness they provide?*

It is important to emphasise that the quality of situational awareness is not solely a measure of how close the agents' belief is to the ground truth. Rather, it is a measure of how well the impact of events and decisions on the mission objectives is understood. To illustrate this important point with an example, consider a disaster management scenario. Within this scenario, it is more desirable to achieve high prediction accuracy within densely populated areas at the cost of having low accuracy in areas with low density, than to have an homogeneously accurate picture of the entire environment.

In what follows, we further qualify and restrict the central challenge, by identifying additional design requirements that good coordination algorithms must satisfy. Then, in Section 1.2 we present our contributions, and discuss how they satisfy these requirements. Finally, we outline the structure of this thesis in Section 1.3.

## 1.1 Design Requirements

The design requirements for a decentralised coordination algorithm can be divided into *functional* and *non-functional* requirements. The former refer to *what* the coordination algorithm should do, while the latter refer to constraints on *how* it should operate. In the discussion above, we have already identified the key functional requirement:

**Quality** The picture that is compiled by the agents should provide high quality situational awareness. This implies that the agents should understand the current state of the world, be able to predict future states, and understand the impact of events and decisions on the mission objectives.

Apart from this functional requirement, we also mentioned two non-functional properties, *robustness* and *autonomy*:

**Robustness** Agents should be able to operate under conditions in which communication is unreliable (for example due to electromagnetic interference or line of sight obstruction), and the failure of a limited number of agents should have little impact on the operation of the remaining functioning ones.

**Autonomy** The responsibility for controlling the agents' actions should lie with the agents themselves. This precludes the use of a centralised controller, whose existence would introduce a single point of failure to the system. Moreover, central control requires a reliable command link between the agents and the controller,

which might not always be present, and might introduce unacceptable delays (e.g. in space exploration) or communication bottlenecks (e.g. in wireless sensor networks). Clearly, in light of these considerations, this requirement is strongly intertwined with that of robustness.

Additionally, we define four more non-functional requirements:

**Adaptiveness** Agents must be able to continuously adapt to their changing environment in order to provide up-to-date situational awareness, and determine what to do next to maintain it. This is important, because many scenarios are characterised by dynamism and inherent uncertainty. Consequently, agents will have limited knowledge of the prevalent conditions before deployment. *Adaptive* agents are capable of continuously revising a model of their environment to best reflect the information that has been gathered so far, and are able to predict and evaluate the outcomes of future decisions.

**Scalability** A coordination algorithm should scale well with the size of a team of agents, both in terms of communication and computational overhead. Specifically, in this thesis, we are interested in decentralised algorithms, the application of which makes the computational overhead incurred by a single agent scale with the number of neighbours, not with the size of the team. This implies that the scale of the team can be virtually unlimited.

**Modularity** A coordination algorithm should be able to coordinate the actions of heterogeneous agents, i.e., agents with different sensing and motion capabilities. This makes it possible to tailor the composition of a team of agents to the specific requirements of the mission (e.g. in a disaster scenario, UAVs can be used to provide a high-level overview of the scene, while UGVs comb the area more thoroughly for survivors). To ensure that different types of agents can cooperate in a single team, the coordination algorithm should not impose restrictions on the implementation of the agents, for example whether the control inputs of the sensors are discrete or continuous, or whether the agents are embodied in fixed sensors or mobile sensors.

**Performance Guarantees** A coordination algorithm should ideally be able to give guarantees on the quality of the situational awareness provided by the agents. Whereas strong empirical results might be sufficient reason to adopt an algorithm in non-critical domains, in sensitive domains such as space exploration it might not, since the existence of pathological behaviour can not be ruled out. Thus, the lack of guarantees can limit an algorithm's applicability in safety critical applications.

Thus far, previous work on sensor networks and mobile robotics has fallen short of providing solutions that exhibit many of these properties. Moreover, many of the previously

developed coordination algorithms are centralised and domain dependent, and thus fail to provide a principled and general approach for controlling a team of autonomous information gathering agents (see Chapter 2 for more details).

In more detail, one influential branch of research focuses on the deployment of fixed or mobile sensors using an *offline* one-shot optimisation phase (Singh et al. 2007, Guestrin et al. 2005, Krause et al. 2006, Zhang & Sukhatme 2007, Meliou et al. 2007), under the assumption that the characteristics of the environment are known beforehand. As a result, these algorithms are not adaptive, and thus are incapable of responding to changing environments or *a priori* unknown events. Moreover, they operate under the assumption that the environment can be considered static during the time the agents require to sample from it. As such, these algorithms fail to model the temporal dynamics of their environment in a principled way; they consider how relevant phenomena vary in space, but not in time. Consequently, the aim of these algorithms is to collect as much information as possible during a single traversal of the space, instead of patrolling it continuously, which is a *sine qua non* of providing accurate situational awareness in dynamic environments. Finally, these algorithms do not model sensors as agents, but rather take a centralised approach. As mentioned earlier, but is worth reiterating, centralisation puts heavy demands on the communication network that exists between the agents, and creates a single point of failure. Thus, these approaches do not meet the requirements of autonomy, adaptiveness, and robustness. As such, they are less suitable in our domain. However, in Chapter 4, we show that particular aspects of these algorithms can be adapted to develop an efficient decentralised algorithm for deploying information gathering agents under additional constraints, and we will discuss these approaches in further detail in Section 2.5.1.

A second major branch focuses on *online* algorithms that control the agents' actions during deployment, instead of planning these actions before deployment (Osborne et al. 2008, Padhy et al. 2010, Kerr & Spears 2005, Pereira et al. 2004, Grocholsky et al. 2006, Martinez-Cantin et al. 2007, Vidal et al. 2001, Singh et al. 2009). As a result, they are often *adaptive*, and are capable of reacting to events which are unknown *a priori*, and are consequently more *robust* against failure. These algorithms can be further categorised in terms of the length of their lookahead. In increasing order of lookahead, these categories are: greedy, finite horizon, and non-myopic. Greedy algorithms, the first category, select the next action only, without regard for their impact on the longer future, and are therefore generally computationally efficient (*scalable*), but can (and often do) converge to poor solutions and lack *quality*. Techniques for greedy control include potential fields (Kerr & Spears 2005, Pereira et al. 2004), and *information surfing* (Grocholsky et al. 2003)— directing the heading of a mobile agent up the 'information gradient'. Finite horizon planning algorithms, the second category, attempt to maximise observation value over an interval that encompasses more than a single action, but is shorter than the remaining mission time (e.g. Martinez-Cantin et al. (2007) and Vidal

et al. (2001)). They generally provide a good trade-off between *quality* and *scalability*, which is why we adopt this approach in Chapter 6. Non-myopic planning algorithms consider the entire (remaining) mission time of the agents. These generally compute high quality solutions, and often give *performance guarantees* (e.g. Singh et al. (2009)). However, this invariably comes at the cost of higher computational overhead due to the large state spaces that need to be searched (for example, using Markov decision processes, Low et al. (2008)), and are often centralised, and therefore fail to satisfy the properties of *autonomy* and *robustness*. Nevertheless, given the possibility of achieving performance guarantees, we shall investigate these methods further in Section 2.5.1, and develop our own non-myopic algorithm with performance guarantees in Chapter 7.

Finally, there is a significant body of research on mobile robotics for target tracking, event detection and map building (e.g. Grocholsky (2002), Calisi et al. (2007), Murphy et al. (2000), Lilienthal et al. (2003), Ahmadi & Stone (2006)). These approaches are robust, modular, and treat each robot as an autonomous agent. However, they are typically tailored to a specific domain, and as such do not provide general solutions that facilitate their application in other information gathering domains.

In the next section we outline how we have addressed these shortcomings in our research.

## 1.2 Research Contributions

The primary aim of the research presented in this thesis is to provide a set of domain independent techniques for coordinating a team of information gathering agents that satisfy the requirements discussed above. To this end, we have formulated a generic model for information gathering with multiple agents (see Chapter 3). This model abstracts from the specifics of an information gathering domain using the notion of *value of observations*, which ranks future observations in terms of their ability to improve situational awareness. We represent this value using a *submodular function* that intuitively captures the diminishing returns of making additional observations. This is somewhat analogous to the work of Meliou et al. (2007) and Singh et al. (2009), except that we explicitly model the temporal dynamics of an environment, in addition to its spacial dynamics. This allows us to develop *domain-independent* algorithms for environments that exhibit a rapid rate of change, and need to be patrolled continuously.

Using this model, we demonstrate the versatility of our approach by studying specific instantiations of it, which include all the examples given at the start of this chapter: monitoring environmental conditions, pursuit evasion (find and capture a moving target), and patrolling (prevent intrusions).

More specifically, we make four main contributions in this thesis. The first two contributions consist of coordination algorithms for *fixed agents* that operate during the

| Chapter | Type of Agent | Planning Horizon | Lifecycle Phase |
|---|---|---|---|
| 4 | Fixed | 1 | Deployment |
| 5 | | | Operation |
| 6 | Mobile | $m$ | |
| 7 | | $\infty$ | |

TABLE 1.1: The roadmap of this thesis.

*deployment* phase (Chapter 4), and the *operation* phase of the agents' life cycle (Chapter 5). The second two pertain to *mobile* agents, for which we develop algorithms with a *receding* planning horizon (Chapter 6), and a *non-myopic* planning horizon (Chapter 7). The roadmap of this thesis shown in Table 1.1 gives a high-level overview of the commonalities and relations between these four contributions. For each contribution, we now briefly highlight their most salient properties in terms of the requirements discussed earlier. These, and the other properties are summarised in Table 1.2.

**Chapter 4. Decentralised Coordination for Fixed Agents during Deployment Phase**

We derive a decentralised algorithm that maximises observation value, while simultaneously constructing a reliable communication network between the agents. Specifically, we present a novel solution to the frequency allocation problem. Instead of solving this NP-hard problem (it is equivalent to graph colouring) for the communication network that exists among all agents directly, the idea is to deactivate certain agents such that the problem can be provably solved in polynomial time. We show that this modified problem is still NP-hard, and develop an efficient approximation algorithm that carefully selects which agents should be deactivated in order to maximise the observation value received by the remaining (active) agents.

We empirically show that this algorithm is no more than 10% away from the optimal solution, and is thus provides high quality situational awareness. Moreover, it is *robust*—it is capable of replacing failed agents with deactivated ones, maintains the *autonomy* of the agents, is very *scalable* (the computational overhead of an agent grows polynomially with the number of neighbours), and we prove that a centralised version of this algorithm provides theoretical *performance guarantees*.

**Chapter 5. Decentralised Coordination for Fixed Agents during Operational Phase**

We develop the first algorithms for distributed constraint optimisation problems (DCOPs) with continuous variables, called CPLF-MS (for linear utility functions)

| | **Chapter** | | | |
|---|---|---|---|---|
| | **4** | **5** | **6** | **7** |
| **Purpose** | Deploying Fixed Agents | Coordinating Fixed Agents | Coordinating Mobile Agents | Coordinating Mobile Agents |
| **Requirement** | | | | |
| Quality | + | + | + | ++ |
| Robustness | + | + | + | 0 |
| Autonomy | + | + | + | 0 |
| Adaptiveness | + | + | + | 0 |
| Scalability | ++ | + | + | 0 |
| Modularity | + | + | + | + |
| Performance Guarantees | + | 0 | 0 | ++ |

TABLE 1.2: An overview of our contributions in terms of the design requirements. The symbols have the following meaning: '+' ('++') means that the requirement is (strongly) satisfied, '0' means the requirement is not satisfied.

and HCMS (for non-linear utility functions). These algorithms are based on the max-sum algorithm (Farinelli, Rogers, Petcu & Jennings 2008), whose applicability was thus far limited to domains with discrete action variables. We study the application of these algorithms on two information gathering settings with fixed agents, and empirically demonstrate their effectiveness.

Specifically, we show that these algorithms respectively improve the solution quality by 10% and 40% in two information gathering domains compared to the standard max-sum algorithm, and therefore improve the quality of situational awareness in these domains. Moreover, they *scale* well, although less so than the algorithm in Chapter 4, since its computational and communication overhead grows exponentially in the number of neighbours of an agents (but not in the total number of agents), and are *adaptive*, since they can be effectively and efficiently run continuously to respond to changes in the agents environments.

**Chapter 6. Decentralised Receding Horizon Control of Mobile Agents**

We develop an adaptive receding horizon control algorithm for mobile agents. Using this algorithm agents, periodically coordinate to maximise the observation value received as a team for a fixed number of time steps $l$ in the future. In more detail, agents coordinate their plans (i.e. finitely long paths in their environment), which they implement for $m \leq l$ time steps. After this, they coordinate again to plan their motion for the next $l$ time steps. Moreover, we assume that this motion is subject to constraints. These motion constraints can be used to model the physical layout of the environment (such as the floor map of a building), as well as the intrinsic movement constraints of the agent itself (e.g. the turning radius of a UAV).

We benchmark this algorithm against both an un-coordinated algorithm and the state-of-the-art (Vidal et al. 2001, Hespanha et al. 1999, Sak et al. 2008) and demonstrate that it increases the *quality* of situational awareness in a variety of highly dynamic domains. In more detail, it increases the quality of the situational awareness of environmental phenomena by up to 50%, decreases the capture time of a target by approximately 30%, and decreases the damage from intrusion by approximately 30%. Furthermore, it is *scalable* since it is based on the max-sum algorithm, so that the computational overhead of a single agent grows with the number of neighbours, not the size of the team; it is *adaptive*, since it is a receding-horizon control algorithm which revises the paths of the agents frequently; and it is *modular*, since it supports agents of different types and corresponding movement constraints. Most importantly, this algorithm is the first online agent-based algorithm for the domains of pursuit-evasion, patrolling, and monitoring environmental phenomena.

**Chapter 7. Decentralised Non-Myopic Control of Mobile Agents**

We present an algorithm for computing *patrols*—infinitely long paths designed to monitor a specific area—in an offline fashion in the same type of environments as the receding horizon control algorithm from the previous chapter. This algorithm follows a similar three-step computation as the algorithm by Singh et al. (2009), i.e., decompose the environment into clusters, compute subpatrols within each cluster, and concatenate these subpatrols to form the desired patrol. However, Singh et al. fail to consider the *temporality* of the environment, which models a continuous rate of change. As a consequence, their algorithm computes finitely long paths, which tend not to return to previously visited locations, since no additional information can be obtained from doing so. In contrast, the patrols computed by our algorithm are designed to monitor continuously changing environments, and thus periodically (and infinitely often) return to the same location to provide up-to-date situational awareness.

Moreover, in contrast to the receding horizon algorithm from the previous chapter, this algorithm is non-myopic, and as such, has an infinite planning horizon. Because of this, the algorithm is able to provide strong *performance guarantees*, at the cost of being less *scalable* than the receding horizon algorithm. The algorithm is a hybrid between offline preprocessing and online decentralised coordination, where the latter is used to improve the *accuracy* of the former, and provide a more *adaptive* solution.

Beside these contributions, this thesis develops several improvements to the max-sum algorithm, which can be regarded as contributions in their own right, and whose applicability goes beyond the scope of the information gathering domain. The max-sum algorithm itself has generated significant attention within the multi-agent community and has been lauded for constituting the middle ground between algorithms that find

optimal solutions at the cost of exponential computation or communication (Modi et al. 2005, Mailler & Lesser 2008, Petcu & Faltings 2005) and algorithms that scale well, but often converge to poor quality solutions (Fitzpatrick & Meertens 2003, Maheswaran et al. 2005, Kiekintveld et al. 2010). In more detail, these contributions are:

- Two general pruning techniques for reducing the computational overhead of the max-sum algorithm. The first technique removes single dominated actions—actions that can never be part of an optimal joint solution. The second technique reduces the size of the joint action space that needs to be searched by the agents. These will be discussed in Chapter 6.
- Two extensions to the max-sum algorithm that allow it to operate on decentralised constraint optimisation problems that are characterised by continuous variables, instead of discrete ones. The first extension derives an exact algorithmic solution for settings where the interactions between variables can be represented by piecewise linear functions. The second extension combines the standard max-sum algorithm with non-linear optimisation techniques for application in non-linear settings. These are presented in Chapter 4.

All the aforementioned contributions lie at the foundation of eight papers:

1. R. Stranders, A. Rogers, and N. R. Jennings. A Decentralised, Online Coordination Mechanism for Monitoring Spatial Phenomena with Mobile Sensors. In *Proceedings of the Second International Workshop on Agent Technology for Sensor Networks (ATSN)*, Estoril, Portugal, 2008, pp. 9–15. See Chapter 6.
2. R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised Coordination of Continuously Valued Control Parameters using the Max-Sum Algorithm. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Budapest, Hungary, 2009, pp. 601–608. See Chapter 5.
3. R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised Coordination of Mobile Sensors using the Max-Sum Algorithm. In *Proceedings of the 21st International Joint Conference on AI (IJCAI)*, Pasadena, USA, 2009, pp. 299–304. See Chapter 6.
4. A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. Decentralised Coordination for Embedded Agents using the Max-Sum Algorithm. Artificial Intelligence Journal (AIJ). Accepted. See Chapter 6.
5. R. Stranders, F. M. Delle Fave, A. Rogers, and N. R. Jennings. A Decentralised Coordination Algorithm for Mobile Sensors. In: *Proceedings of the Twenty-Fourth*

*AAAI Conference on Artificial Intelligence (AAAI)*, Atlanta, USA, 2010, pp. 874–880. See Chapter 6.

6. R. Stranders, A. Rogers, and N. R. Jennings. A Decentralised Coordination Algorithm for Maximising Sensor Coverage in Large Sensor Networks. In: *Proceedings of the Ninth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Toronto, Canada, 2010, pp. 1165–1172. BAE Systems, one of the sponsors of the ALADDIN project, has applied for a patent on the algorithms in this paper (British Patent application reference GB1001732.5). See Chapter 4.

7. T. Voice, R. Stranders, A. Rogers, and N. R. Jennings. A Hybrid Continuous Max-Sum Algorithm for Decentralised Coordination. In *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI)*, Lisbon, Portugal, 2010, pp. 61–66. *Shortlisted for best paper award.* See Chapter 5.

8. R. Stranders, E. Munoz de Cote, A. Rogers, and N. R. Jennings. Non-Myopic Bounded Approximation for Infinite Horizon Patrolling with Mobile Sensors. Artificial Intelligence Journal (AIJ). In preparation. See Chapter 7.

## 1.3 Thesis Structure

The remainder of this thesis is organised as follows:

- In Chapter 2, we discuss related work. We propose a generic architecture for an information gathering agent in order to analyse the state of the art. We review the techniques that have been employed in each of the components of the architecture, and identify the key methods that have been adopted in our research.

- In Chapter 3 we present a generic model for the problem of information gathering with multiple agents. We define the key concept of *observation value* that is strongly related to the *quality* of the situational awareness the agents provide, and express the objective of the agents in terms of this concept.

- In Chapter 4 we present a decentralised coordination algorithm for deploying a network of fixed information gathering agents. This algorithm maximises observation value, while simultaneously building a reliable communication network between the agents.

- In Chapter 5 we investigate the use of decentralised coordination for fixed agents after their initial deployment. We study two information gathering domains in which the agents' control parameters are continuous, and we present and apply two extensions to the standard max-sum algorithm that allow it to solve continuous decentralised optimisation problems.

- In Chapter 6, we turn to the challenge of coordinating mobile agents. We show how the max-sum algorithm can be applied to coordinate the agents' movements in order to maximise their joint performance subject to movement constraints, and present two generic techniques for reducing its computational overhead.
- In Chapter 7, we develop a coordination algorithm for mobile agents with performance guarantees. This algorithm computes infinitely long patrols for continuously monitoring dynamic environments in an offline phase. We then show how online decentralised coordination can be used to improve the quality of these patrols, and to adaptively respond to a priori unknown events.
- Finally, in Chapter 8 we conclude and present directions for future work to broaden the scope of our research and increase its practical applicability to autonomous control of unmanned sensors.

# Chapter 2

# Literature Review

The literature on information gathering systems (agent-based, centralised or otherwise) is rich and diverse. It encompasses the use of sensors in a multitude of application domains, ranging from agriculture, military surveillance, disaster management, and climate research. Within each of these domains, these sensors are embedded in yet another multitude of different types of hardware, which are either mobile (UAVs and UGVs), or fixed (e.g. sensor nodes in a wireless sensor network). In order to navigate through and make sense of this large body of research—in particular in the context of this thesis—we will analyse the state of the art by providing an 'exploded' view of existing approaches in information gathering, which shows the relationship between their constituting parts. To assist in this analysis, we propose a general architecture of an information gathering system in Section 2.2. This architecture consists of the three essential components of an information gathering system, which correspond to the three central challenges it needs to address:

1. Representing the environment: how are observations from the environment processed to obtain a high-level representation of the environment? (Section 2.3)
2. Valuing observations: how should observations be ranked in terms of their contribution towards more accurate situational awareness? (Section 2.4)
3. Coordination: how should the agents be controlled in order to maximise observation value, subject to movement and temporal constraints? (Section 2.5)

Now, a large portion of the state of the art is tailored to specific domains, and thus solves these three challenges in domain dependent ways. However, by decomposing an information gathering system in this fashion, it becomes possible to analyse the third component, coordination, in a domain independent fashion. Whilst the component responsible for representing the environment transforms inherently domain dependent observations into an abstract model of the environment, and the observation value component depends on

this model for ranking future observations, the coordination component is fully shielded from domain dependent features through the domain independent concept of *observation value*. The main advantage of this is that it allows us to ascertain the strengths and weaknesses of coordination algorithms in a domain independent fashion, and adapt them to suit our purpose.

To further facilitate analysis, we categorise these coordination algorithms into offline and online algorithms, which perform their computation in different phases: the former pre-compute coordinated plans for the agents before their deployment, while the latter controls the agents during their operation. While offline are better suited and more efficient for deploying agents if it is known beforehand which features of the environment should be monitored, online algorithms can generally better adapt to more uncertain environments, and are more robust to failure—two desirable properties in terms of the requirements discussed in Chapter 1. Decentralised coordination techniques, a subclass of online algorithms (which also contains centralised online algorithms), will receive special attention in Section 2.6, given their importance to the main challenge of this thesis.

Before commencing our analysis of the technical aspects of information gathering systems, however, we shall first briefly introduce three exemplar information gathering domains that will be used throughout this thesis, and will serve to illustrate the analysis offered in this chapter.

## 2.1 Exemplar Information Gathering Domains

Information gathering is an abstract name for a set of real-life and important applications. In order to make the upcoming analysis more concrete, and to be able to discuss application specific techniques, we introduce three instances of information gathering domains: monitoring environmental phenomena, wide area surveillance and pursuit evasion. These three domains should by no means be regarded as an exhaustive enumeration of the set in which they are contained. Instead, they are chosen to illustrate they breadth and richness of this set, and to show that the literature often treats them as completely unrelated subjects. As we shall see in Chapter 3, this thesis unifies these seemingly different problems into a single general problem formulation, making it possible to solve them with general and principled algorithms.

### 2.1.1 Monitoring Environmental Phenomena

An environmental phenomenon is a real valued field over up to three spatial dimensions, and possibly a temporal dimension. Examples of environmental phenomena include radiation, temperature, pressure, gas concentration, pH value, humidity, tidal height

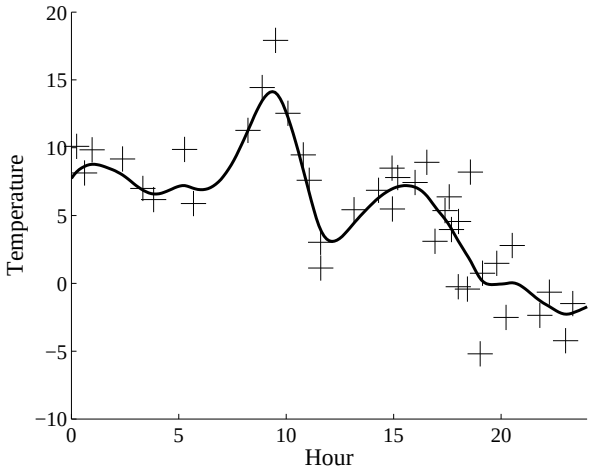



FIGURE 2.1: An environmental phenomenon (temperature) with noisy observations measured over time at a fixed point in space.

and wind speed. These phenomena play an important role in climate research (Padhy et al. 2010, Hart et al. 2006), agriculture (Zhang 2004, Edordu & Sacks 2006, Galmes 2006, Langendoen et al. 2006, Wark et al. 2007), local weather and tide predictions (Kho et al. 2009, Osborne et al. 2008), environmental control of "intelligent buildings" (Deshpande, Guestrin & Madden 2005), and gas source tracing (Kato & Mukai 2005, Lilienthal et al. 2003).

Agents in this domain are tasked with constructing an accurate model of these phenomena, based on (possibly noisy) observations. An example of a (simple) model is shown in Figure 2.1, where regression is applied to a few temperature measurements in order to estimate the underlying dynamics of the phenomenon. This particular example has only one dimension (a temporal one), but could easily be extended with three spatial dimensions. In fact, in the literature, virtually all possible combinations of spatio-temporal dimensions are addressed. To give two concrete examples, the Networked Infomechanical System (NIMS) (Rahimi et al. 2004, Pon et al. 2005) is a single sensor suspended on a wire, that can move horizontally and vertically to take pH measurements in a cross-section of a river, and thus operates in two spatial dimensions; and Floodnet, a flood warning system, which consists of multiple fixed sensors each of which samples along the temporal dimension, but combines their readings into a spatio-temporal picture. We come back to modelling and representing environmental phenomena in Section 2.3.1.

### 2.1.2 Wide Area Surveillance

Wide area surveillance is an umbrella term for the passive detection, classification and tracking of events. Within each of these scenarios, agents are equipped with cameras, radars, or motion detectors. These agents are usually in some way resource constrained. For example, they have a limited battery life such that they have to carefully schedule

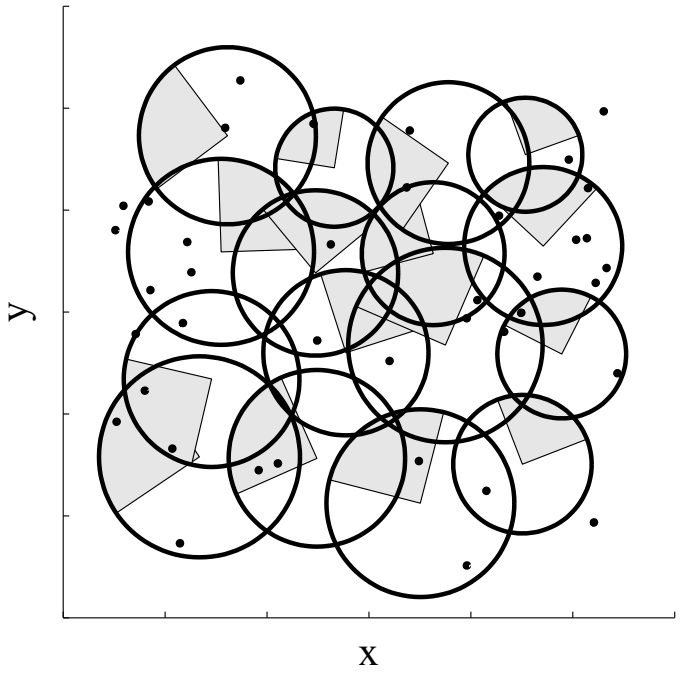



FIGURE 2.2: An example of a wide-area surveillance scenario with 16 agents, and 40 events (dots). The agents are capable of directing their sensors to observe a segment (grey areas) of their maximum sensing radius (circles).

their sensing intervals (Farinelli, Rogers & Jennings 2008), or can only observe a fraction of their sensing area at a time (Kim et al. 2010, Dang et al. 2006). An example of the latter is shown in Figure 2.2, in which agents adjust their viewing angle to observe a segment within their observation radius. Within this segment, agents are able to detect or classify targets, depending on the application. In Section 2.3.2, we go into further detail on representing wide area surveillance scenarios.

### 2.1.3 Search and Patrol

Search and patrol is a class of problem that is commonly found in disaster management and security domains, where, for example, agents have to search for wounded civilians or prevent intrusions of a perimeter. In these domains, agents are tasked with gathering information about the (possible) location of wounded or attackers, with the aim of minimising their detection time. In this thesis, we study two concrete instances of search and patrol: pursuit evasion and patrolling.

Pursuit evasion is characterised by the presence of an evader that the agents need to capture as quickly as possible (Bopardikar et al. 2008, Hespanha et al. 1999, Vidal et al. 2001, Borie et al. 2009, Halvorson et al. 2009). This evader might want to be found (such as a confused civilian in a disaster scenario) or it might not (such as an intruder in a security scenario). Moreover, the environment in which both the agents and the evader

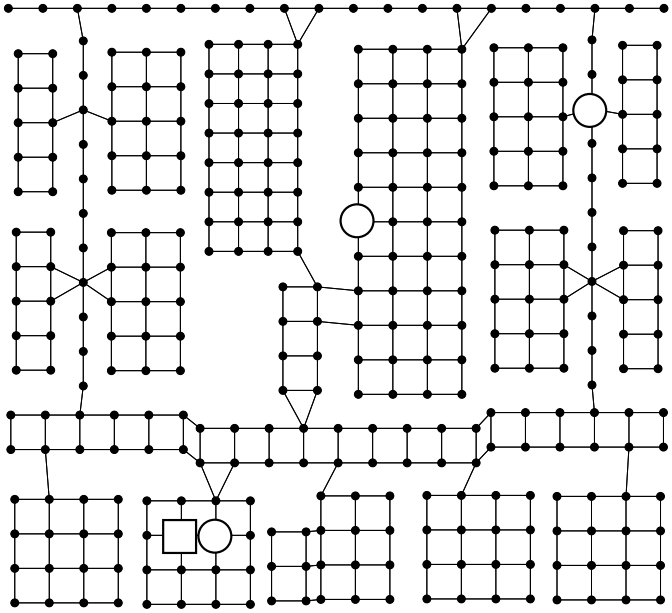


FIGURE 2.3: An example of a pursuit evasion scenario with three agents (circles) and an evader (square) in a layout graph of connected locations. The evader is within capture range of one of the agents

exist can be bounded (such as a building) or unbounded (such as airspace). Although they might seem very similar, these two types of environments require significantly different representations and techniques. In this thesis, we shall primarily focus on bounded pursuit evasion scenarios, in which the agents' motion is constrained by the layout of the environment. An example of such a scenario is shown in Figure 2.3, where three agents are pursuing a single evader in the Intelligence, Agents, Multimedia (IAM) lab of the School of Electronics and Computer Science.

The second instance of the search and patrol domain we consider in this thesis is patrolling, in which (possibly multiple) attackers attempt to intrude into the environment (Agmon, Kraus & Kaminka 2008, Paruchuri et al. 2007, Basilico et al. 2009), for example via a perimeter (and are thus not already present in the environment as in the pursuit evasion scenario). Once they succeed, the agents incur an immediate loss. Thus, the agents' main challenge is to detect and thwart these intruders so as to minimise this loss.

In Section 2.3.3, we will discuss techniques for representing both instances in more detail.

## 2.2 A General Architecture for Information Gathering Systems

In the previous section, we gave three concrete examples of applications of information gathering. In this section, we shift our attention from the application dimension to the

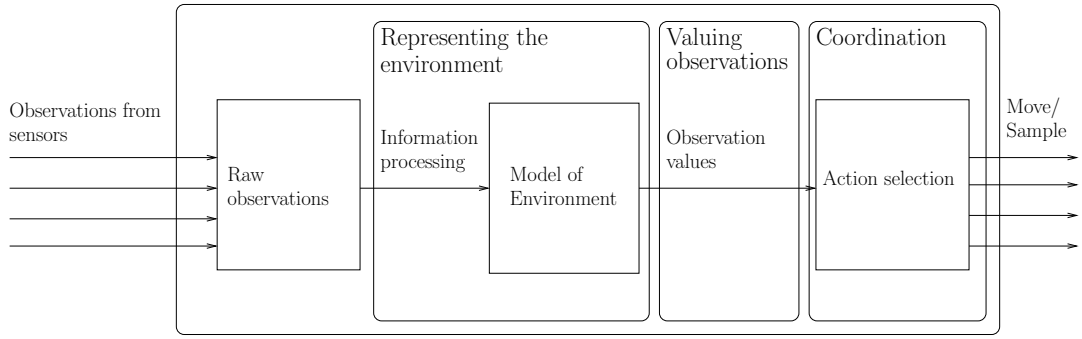



FIGURE 2.4: A general architecture of an information gathering system.

technical dimension, and investigate *how* an information gathering system can accomplish its goals. We specifically use the term information gathering *system* here, instead of information gathering *agent*, since the state of the art in this field of research is not limited to agent-based systems. Moreover, at this point, it is important to include all techniques that could lead to key insights into developing decentralised coordination algorithms that satisfy many of the requirements stated in Chapter 1.

Now, the operation of information gathering systems can be described as performing *adaptive sampling* (Kho et al. 2009, Zhang & Sukhatme 2007, Zhou et al. 2006, Osborne et al. 2008). Informally, adaptive sampling can be thought of as "intelligent sampling"; exploiting the specific properties of the environment in order to maximise observation value, subject to the limited resources of the agents (e.g. battery power and communication) and the constraints imposed by the environment (such as obstructions). To illustrate the need for adaptive sampling, consider a very naïve approach that samples at a fixed rate. Generally speaking, in most dynamic environments the rate of change is not constant; it varies over space and time. Thus, in these environments, fixed rate sampling will observe the agent's surroundings equally often at times in which the environment is changing rapidly, as at times when it is fairly static, resulting in suboptimal observation value. So, instead, an information gathering system needs to carefully decide when and where to make observations in order to best allocate its resources. Given these considerations, we can define adaptive sampling as:

**Definition 2.1** (Adaptive Sampling)**.** Making observations in space, time or both, so as to maximise observation value obtained by one of more information gathering agents, (possibly) subject to movement constraints and scarce resources such as communication bandwidth and time.

Adaptive sampling in the literature is performed in a variety of ways, with an even larger variety of techniques. The tool that will aid us in organising and analysing these techniques is a general architecture of an information gathering system. This architecture consists of the three main components, each of which addresses one of the

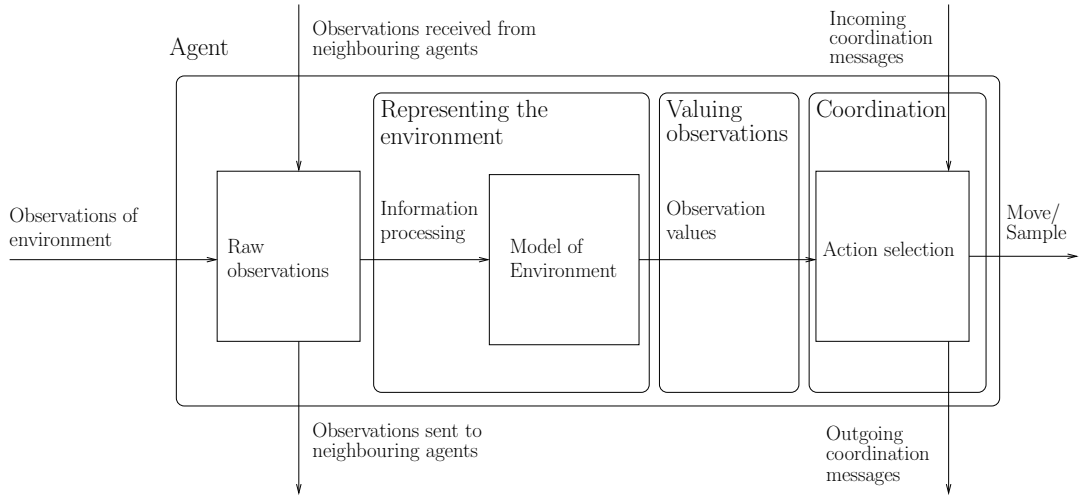



FIGURE 2.5: A general architecture of an information gathering agent.

sub-challenges of adaptive sampling. These were already mentioned in the introduction to this chapter, but are repeated here in some more detail:

1. Representing the environment. Observations collected by the sensors are processed into an abstract model of the environment.
2. Valuing observations. Future observations are valued (ranked) in terms of their potential to increase the quality of situational awareness.
3. Coordination. The system coordinates the actions of the sensors, and determines which observations to make so as to maximise the value of these observations.

For a *centralised* information gathering system, this architecture is shown in Figure 2.4. Such a system receives observations from multiple sensors (left), goes through the three steps mentioned above, and communicates the computed control inputs back to the sensors (right).

Instead of having a centralised controller, an *agent-based* information gathering system is characterised by the distribution of control over multiple agents. The architecture for a *single* agent is shown in Figure 2.4. A multi-agent system is obtained by allowing agents to communicate with their neighbours, through the exchange of observations and coordination messages (the vertical arrows).

Now, in contrast to a centralised system, a single agent's world view is limited to the observations it makes itself, and those communicated to it by its neighbours. However, since these observations can be propagated through multi-hop communication (possibly by piggybacking on coordination messages), agents can potentially achieve a high level of belief synchronisation (Makarenko & Durrant-Whyte 2006). The observations

obtained in this fashion are processed by the first two components of the agent, which are responsible for representing the environment and valuing observations—similar to the centralised system. However, the third component—coordination, which is the main focus of this thesis—differs from that of a centralised system, since agents do not have a global view of the effect of their collective actions. As a result, without coordination with their neighbours, the agents' actions often lead to suboptimal team performance (due to redundant coverage of some areas, no coverage of other areas, collisions, etc.), which results in low *quality* of the achieved situational awareness. Thus, agents need to coordinate by communicating with their neighbours in order to maximise the observation value received as a team. By so doing, the agents establish *decentralised coordination*, which yields coordinated joint actions.

In Section 2.6 we come back to this important topic of decentralised coordination. First, however, we discuss each of the three aforementioned components in more detail, starting with representing the environment.

## 2.3 Representing the Environment

In this section, we focus our attention on the first component of the architecture in Figures 2.4 (for centralised systems) and 2.5 (for agent-based systems), which is responsible for transforming a collection of raw observations into a representation, or model, of the environment.

In order to choose an appropriate representation technique, we first need to identify the type of environment within which the agents exist. More specifically, the type of *change* the environment is subjected to—temporal, spatial or both—is of significant importance, since it determines to great extent the type of representation that is required to accurately capture the spatial and temporal dynamics of the environment. These dynamics are essential to recover the current state of the environment, as well as predict its future states—the two key requirements of situational awareness.

In more detail, we can distinguish three types of adaptive sampling algorithms, corresponding to three types of environment:

**Spatial adaptive sampling** These algorithms operate in environments that are not changing (or changing negligibly) over time. Generally speaking, spatial adaptive sampling algorithms fall into two different and distinct classes. The first consists of algorithms for computing informative placements for fixed sensors, given that the spatial dynamics of the environment are known beforehand (Guestrin et al. 2005, Krause et al. 2006).

The second class consists of algorithms for observing environments with mobile robots. These environments are either static, or change so slowly that they can

be considered static during the time it takes to traverse them (Rahimi et al. 2004, Pon et al. 2005, Krause & Guestrin 2007, Singh et al. 2009, Zhang & Sukhatme 2007). As a consequence, these algorithms compute finitely long paths, which tend not to return to previously visited locations, since no additional information (or value) can be obtained from doing so. These paths are used to traverse the space once; agents attempt to take the most informative observations, after which they return to a base station.

**Temporal adaptive sampling** These algorithms are mainly used for controlling fixed agents that can only choose their observations along the temporal dimension. These agents are deployed in environments where change occurs over time. Within the exemplar domains from Section 2.1, the *rate* of change can vary significantly. Examples of low rates of change are found in environmental monitoring where the deployment time is sufficiently long to detect the effect of day-night cycles, such as temperature and tidal heights (Kho et al. 2009, Osborne et al. 2008) or, on a longer time-scale, seasonal change (Padhy et al. 2010). Notable examples of domains governed by high rate of changes include military surveillance (e.g. target tracking or intrusion detection) and disaster management (Waldock et al. 2008, Dang et al. 2006). In both cases, samples must be carefully scheduled over time in order to sample efficiently. Sampling at a constant rate wastes resources in periods where the environment is changing slowly, and provides insufficient resolution in intervals of rapid change.

**Spatio-temporal adaptive sampling** The class of spatio-temporal algorithms sample along the temporal dimension, and at least one spatial dimension. These algorithms are commonly found in environments that change at a rapid rate, or in environments in which mobile agents patrol continuously. Examples of the former include the use of fixed sensors for weather prediction, where multiple fixed sensors are used to recover the spatio-temporal correlations that govern the surrounding (micro) climate (Osborne et al. 2008). In the latter case, even if the environment is changing slowly, conditions might have changed significantly by the time the agents have traversed the space. Examples of this include gas source tracking (Kato & Mukai 2005, Lilienthal et al. 2003, Zhang et al. 2005), and target tracking (Grocholsky 2002, Grocholsky et al. 2006, Makarenko & Durrant-Whyte 2006). In either case, time is a variable that cannot be neglected. This means that agents need to be able to determine how to sample in space-time in order to be able to reconstruct the dynamics of the underlying phenomenon.

It is important to note that, in contrast to spatial adaptive sampling, in spatio-temporal adaptive sampling it is generally prudent to return to previously visited locations. As said earlier, by doing so, no new information can be obtained in environments that are *static* over time. However, in environments that *do* change over

time, continuous patrolling is essential for achieving accurate situational awareness. Given the prevalence of dynamic environments in the domains we consider in this thesis (i.e. the exemplar domains from Section 2.1), we develop algorithms for continuous patrolling in Chapters 6 and 7.

Thus, in order to achieve situational awareness in a dynamic environment, it is vital that neither its temporal, nor the its spatial dynamics can be ignored. On the one hand, an algorithm that focuses solely on spatial correlations might be able to give an accurate picture of the current state of the world, but not on what will happen in the near future. On the other hand, an algorithm that is able to predict what will happen in only a small area of the environment provides an incomplete spatial picture of the world. Clearly, making decisions based on either spatially or temporally deficient situational awareness exposes the decision maker to risks.

In light of this, in Chapter 3, we explicitly model the spatial and temporal dynamics of an environment, and develop algorithms in Chapters 4, 5, 6, and 7 that operate directly on this model, thus achieving spatio-temporal adaptive sampling.

Now that we have classified adaptive sampling in terms of the spatio-temporal dimensions that are taken into consideration, we now turn to the domain specific techniques that are used for representing the environment in the three exemplar domains of Section 2.1: monitoring environmental phenomena, wide area surveillance, and pursuit-evasion.

### 2.3.1 Monitoring Environmental Phenomena

The main challenge in monitoring environmental phenomena is identifying spatial and temporal patterns in the (possibly) noisy observations that have been made. As discussed earlier, these patterns are required to predict unobserved locations, as well as the future state of the world. Regression is commonly used to accomplish this, two types of which we will discuss in this section: linear regression (Section 2.3.1.1), and the more expressive and flexible Gaussian process regression (Section 2.3.1.2). The latter will receive some more in-depth coverage, since it is our tool of choice for modelling these phenomena (see Chapters 6 and 7). Finally, in Section 2.3.1.3, we discuss techniques that do not fall in either category, but which are, nevertheless, relevant to our work.

#### 2.3.1.1 Linear Regression

Linear regression models a phenomenon by a collection of linear relations between the explanatory variables (such as time and location) and the variable of interest (such as temperature or gas concentration). Its attractiveness stems from its simplicity and

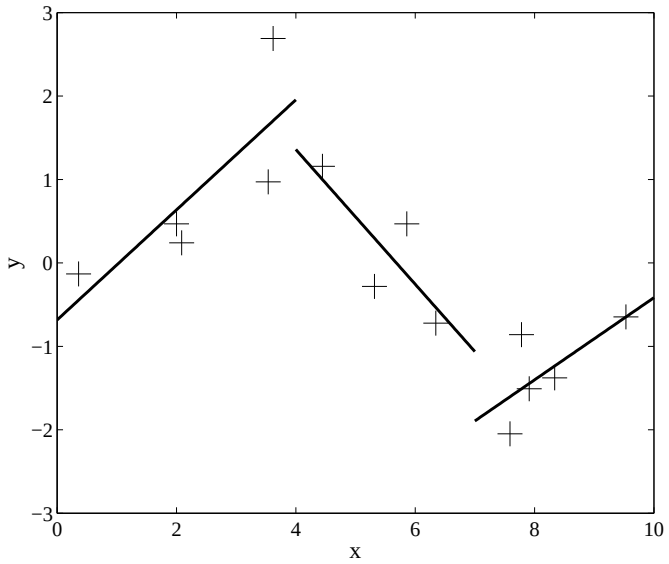



FIGURE 2.6: The piecewise linear regression method proposed by Padhy et al. (2010) applied to an environmental phenomena with non-linear correlations. In this example, the non-linear phenomenon is approximated by three line segments.

computational efficiency. For this reason, its use has been suggested for resource constrained sensing agents in Floodnet, a flood monitoring system (Kho et al. 2009), and for environmental monitoring (Padhy et al. 2010). In the latter case, the environmental variables (i.e. temperature and pressure) are characterised by their non-linear relation with time. Consequently, these relations are modelled by piecewise linear functions, whereby Bayesian inference is used to decide whether the newly obtained sample can be sufficiently explained by the current regression model, or whether the sample represents a change point and the model needs to be discarded in favour of a new one (see Figure 2.6 for an example).

In the two-dimensional (spatial) case, Zhang & Sukhatme (2007) employ linear regression to model temperature variations in a body of water, where temperature measurements are expected to be inversely correlated to the distance between the locations at which they were taken.

Although linear regression is attractive due to its simplicity and its low computational cost, it lacks universal applicability, since many real-life phenomena are governed by (strongly) non-linear relations. Moreover, to ensure the generality of our algorithms, we do not wish a regression method to restrict the class of possible applications, even if this results in less efficient algorithms. Thus, we believe that we should allow for a more flexible type of regression. With this in mind, we will now turn to the Gaussian process, which is a more flexible, and therefore more suitable alternative.

#### 2.3.1.2 Gaussian Processes

Gaussian processes (GPs) provide a very flexible way of performing regression, because, unlike linear regression, they are not limited to (piecewise) linear functions. In what follows, we will offer a basic introduction to the Gaussian process.[1] First, we will explain its basic properties, and describe the role played by the covariance function, and how a GP is fitted to observations made. Then, we discuss the use of the GP in related work, and explain how it will be used in our work.

Before we continue with the specifics of the GP, we will define some of the notation. Let the process $y = f(\mathbf{x})$ denote the relation between a $D$-dimensional input vector $\mathbf{x} \in \mathbb{R}^D$, and an output variable $y \in \mathbb{R}$. Moreover, let $\{(\mathbf{x}_i, y_i) \mid i = 1 \dots n\}$ denote a set of input-output pairs, which represent the observations of this process $f$, and is commonly referred to as the training set. Furthermore, variables subscripted with a star (such as $\mathbf{x}_*$) denote predictions (or test data). For example, $y_*$ denotes the predicted function value for input vector $\mathbf{x}_*$. Finally, we denote the $n$-dimensional vector of all collected outputs $y_i$ as $\mathbf{y}$, and collect the $n$ inputs $\mathbf{x}_i$ in a $D \times n$ matrix $X$.

Now, Rasmussen & Williams (2006) define a GP as:

**Definition 2.2** (Gaussian process)**.** A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.

It is well-known that a finite set of jointly Gaussian random variables is fully determined by their mean $\boldsymbol{\mu}$ and covariance matrix $\Sigma$. Now, given this fact, we can generalise these definitions of mean and (co)variance to an infinite set of variables, and fully determine a Gaussian process by a mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$:

$$\begin{aligned} m(\mathbf{x}) &= E[f(\mathbf{x})] \\ k(\mathbf{x}, \mathbf{x}') &= E[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \end{aligned}$$

where $E[X]$ is the expectation of random variable $X$. Here, covariance function $k$ determines the covariance between two outputs of $f$ as a function of their associated inputs $\mathbf{x}$ and $\mathbf{x}'$. Generally, this is a decreasing function of the distance between $\mathbf{x}$ and $\mathbf{x}'$. Now, if both $m(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x}')$ are given, they function as a prior over the function $f$. This prior is based on properties of $f$ that are known *a priori*, such as smoothness and rate of change. It functions as a probability distribution over possible functions, if no evaluations of $f$ are available. However, if training data $\{(\mathbf{x}_i, y_i) \mid i = 1 \dots n\}$ (i.e. observations of function $f$) are available, a GP can be fitted to these data, thereby increasing the accuracy of predicting $\mathbf{y}_*$ (the test data) at unobserved locations. In order to do this, we exploit the fact that the prior joint distribution of $[\mathbf{y}, \mathbf{y}_*]^\top$ is multivariate Gaussian:

[1] The following technical description of the Gaussian process is adapted from Rasmussen & Williams (2006), Section 2.2.

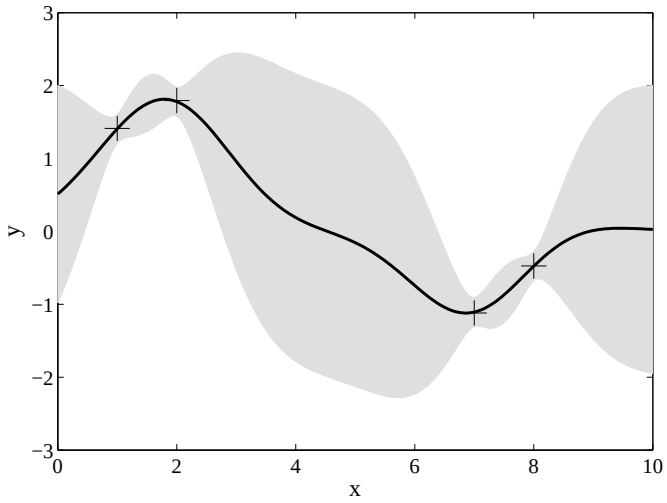



FIGURE 2.7: Fitting a GP to data. The grey area represents the 95% confidence band (two standard errors) derived from covariance matrix $\Sigma$.

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right)$$

where the matrices $K(\cdot,\cdot)$ are obtained by evaluating the covariance function $k$ for all pairs of columns of the offered matrices. Thus, $K(X,X)$ is the covariance matrix for all pairs of training points, and the $K(X,X_*)$ is the covariance matrix for all pairs of training and test points. The posterior distribution of $\mathbf{y}_*$ given $\mathbf{y}$ can now be easily obtained from Bayes' theorem, using the properties of the Gaussian distribution:

$$P(\mathbf{y}_* \mid X_*, X, \mathbf{y}) = \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

where mean vector $\boldsymbol{\mu}$, and covariance matrix $\Sigma$ are given by:

$$\boldsymbol{\mu} = K(X_*,X)K(X,X)^{-1}\mathbf{y} \tag{2.1}$$

$$\Sigma = K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*) \tag{2.2}$$

Figure 2.7 shows an example of a GP fitted to four observations.

Here, it is important to note that the covariance of $\mathbf{y}_*$ does not depend on any actual observations $\mathbf{y}$, but only on their input vectors $X$. Put differently, if the covariance function is known, the covariance of the predictions depends only on the *locations* where the observations were taken, not on their actual *values*. As we will see in Section 2.4, this is a property of GPs that is widely exploited by algorithms that calculate informative placements or paths for fixed and mobile agents, without the need for any

prior sampling, and can lead to an intuitive formulation of the value of observations in the second component of the architecture of an information gathering agent.

In the Gaussian process, the covariance function $k(\cdot,\cdot)$ plays a critical role; it determines the covariance between two sample points of the process $f$ and thus it restricts the class of functions over which the Gaussian process performs regression. A typical choice of a covariance function is the squared exponential covariance function. This function models the correlations between two sample points for a large class of smooth, non-linear functions. Within this class, the covariance between two points is inversely proportional to their distance. In its simplest form, the function is defined as:

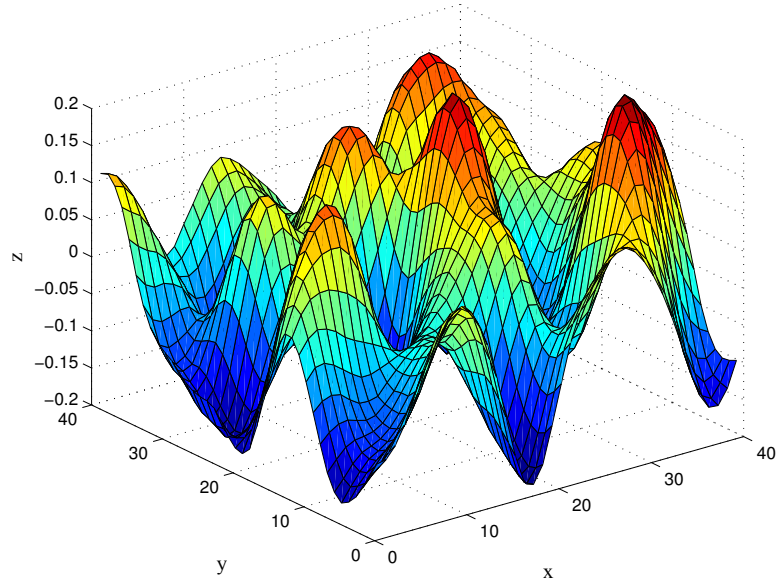
$$k(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{1}{2}\left|\mathbf{x}-\mathbf{x}'\right|^2\right) \tag{2.3}$$

The use of this covariance function leads to a very smooth process. Moreover, this process will exhibit the same characteristics over each of its input dimensions. That is, the process is insensitive to translation and rotation. Such a process is called *stationary*. To model *non-stationary* processes, a transformation matrix $\mathbf{P} = diag(l_1^2,\ldots,l_D^2)$ is introduced, and $|\mathbf{x}-\mathbf{x}'|$ is replaced by $(\mathbf{x}-\mathbf{x}')^\top\mathbf{P}^{-1}(\mathbf{x}-\mathbf{x}')$. The entries $l_1^2$,...,$l_D^2$, scale the dimensions of the input vector $\mathbf{x}$ independently.[2] Depending on the type of dimension to which they apply, these entries are more commonly referred to as *length-scales* or *time-scales*. Generally, the more gradually the modelled phenomenon varies over an input dimension, the longer length-scale for that dimension, and vice versa. To put this in the context of environmental phenomena, these scales allow us to model processes that are strongly correlated along one input dimension, while weakly correlated along another. To illustrate this, Figure 2.8 shows an example of the effect of varying the two length-scales in a two-dimensional process.

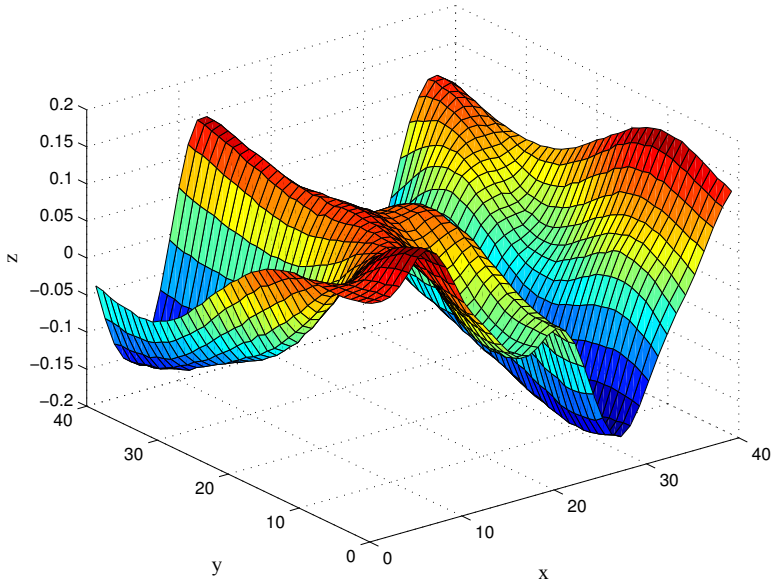
Thus far, we have assumed noise-free observations. However, in many practical sensor deployments observation noise cannot be neglected. In this case, the sensors do not observe the process $f(\mathbf{x})$ itself, but a noisy version of it: $f(\mathbf{x})+\varepsilon$. Assuming that $\varepsilon$ has a Gaussian distribution with variance $\sigma_n^2$ and is independent of the input $\mathbf{x}$, we add an extra term $\sigma_n\delta_{\mathbf{x}\mathbf{x}'}$ to the covariance function, where $\delta_{\mathbf{x}\mathbf{x}'}$ is the Kronecker delta which equals one iff $\mathbf{x}=\mathbf{x}'$. To see why this models noise, note that this term adds $\sigma_n^2$ to the diagonal of the covariance matrix, effectively increasing the variance of the associated output-variable. For example, extending the squared exponential covariance function in Equation 2.3 for noisy observations, we obtain:

$$k(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{1}{2}\left|\mathbf{x}-\mathbf{x}'\right|^2\right) + \sigma_n^2\delta_{\mathbf{x}\mathbf{x}'} \tag{2.4}$$

[2]Note that for $l_i = 1$ for $1 \le i \le D$, the numerator of the exponent becomes the square of the Euclidean distance between the two vectors, in which case the resulting GP exhibits the same characteristics over all input dimensions.




(a) A function drawn from a process with equal length-scales along both dimensions.




(b) A function drawn from a process with a longer length-scale along the $y$ dimension than along the $x$ dimension. Note that the process now varies less rapidly along the $y$ dimension.

FIGURE 2.8: Two bi-variate functions drawn from GPs with a squared exponential covariance function, showing the effect of varying the length-scales.

**Algorithm 1** Prediction of values at coordinates $X_*$ given training data $X$, $\mathbf{y}$ and covariance function $k$ using the Gaussian process (Adapted from Rasmussen & Williams (2006), Algorithm 2.1).

1: **Input:** matrix with training inputs $X$, training outputs $\mathbf{y}$, covariance function $k$, noise $\sigma_n^2$, $X_*$ test inputs. $K(X, X')$ denotes the matrix obtained by evaluating the covariance function $k$ for every pair of columns of $X$ and $X'$.
2: **Output:** $(\boldsymbol{\mu}, \Sigma)$ such that $\mathbf{y}* \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$
3: $L \leftarrow cholesky(K(X, X) + \sigma_n^2 I)$
4: $\boldsymbol{\alpha} \leftarrow L^\top \setminus (L \setminus \mathbf{y})$
5: $\boldsymbol{\mu} \leftarrow K(X_*, X_*)^\top \boldsymbol{\alpha}$
6: $V \leftarrow L \setminus K(X, X)$
7: $\Sigma \leftarrow K(X_*, X_*) - V^\top V$

Now, at the start of this section, we argued that modelling both both spatial and temporal correlations of a phenomenon is crucial for achieving good situational awareness. The GP allows us to create processes that have different correlation structures along different dimensions. In its most general form, a process with two spatial and one temporal dimension, where input vector $\mathbf{x} = [x \quad y \quad t]^\top$, has a covariance function $k(\cdot, \cdot)$ that is the product of a spatial covariance function $k_s(\cdot, \cdot)$, and a temporal covariance function $k_t(\cdot, \cdot)$:

$$k\left(\begin{bmatrix} x \\ y \\ t \end{bmatrix}, \begin{bmatrix} x' \\ y' \\ t' \end{bmatrix}\right) = k_s\left(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} x' \\ y' \end{bmatrix}\right) \cdot k_t\left(t, t'\right) \tag{2.5}$$

For example, an environmental phenomenon that exhibits smooth change over space, but less-smooth change over time, can be modelled by choosing a squared exponential covariance function for the spatial dimension, and a Matérn covariance function[3] for the temporal dimension. The input-scales of this process are modelled by the $\mathbf{P}$ matrices of the two covariance functions that compose it. For the spatial covariance function $\mathbf{P} = diag(l_s^2, l_s^2)$, with entries that determine the length-scale of the process, and for the temporal covariance function $\mathbf{P} = l_t^2$, containing a single entry encoding the time-scale of the process. For example, an environmental phenomena that varies slowly over space, but very quickly over time has a long length-scale $l_s^2$, but a short time-scale $l_t^2$.

Now that we have discussed some examples of covariance functions and their properties, only the question of fitting a GP to data remains unanswered. Using Equations 2.1 and 2.2, calculating predictions for arbitrary test inputs $\mathbf{y}_*$ is fairly straightforward. However, the explicit inversion of the covariance matrix $K(X, X)$ as prescribed is very computationally intensive,[4] especially if the training set is very large. Instead, there exists a more efficient method that exploits the fact that the covariance matrix $\Sigma$ is symmetric

[3] This is a second often-encountered class of covariance functions that is more suitable for modelling less-smooth processes.

[4] For a $n \times n$ matrix $K(X, X)$, explicitly calculating $K(X, X)^{-1}$ takes $n^3/3$ operations with Gauss-Jordan elimination.

and positive definite. This allows for the matrix to be decomposed into a product of a lower triangular matrix $L$ and its transpose: $\Sigma = LL^\top$, known as a Cholesky decomposition. Algorithm 1 efficiently computes the mean and variance of a set of test inputs by exploiting this decomposition.[5] However, computing the Cholesky decomposition can lead to an unacceptable overhead, particularly if new data points arrive sequentially. In order to overcome this potential bottleneck, Osborne et al. (2008) present a number of efficient numerical algorithms for updating the Cholesky decomposition with newly acquired data points, as well as down dating it after discarding old data points in order to ensure that the size of the matrix $L$ stays within reasonable bounds. This not only saves memory and computational resources, but also ensures that predictions are based on the most recent (and relevant) part of the observation history.

Until now, we have assumed that the covariance function is fully known. In most realistic scenarios, however, the covariance function has to be learnt from data. In these cases, the use of GPs can be regarded as a three-level model selection problem. On the top level, the shape of the covariance function determines the high-level properties of the process. More specifically, it determines its smoothness (or differentiability), whether the process is periodic or isotropic. On the second level, the variables of the particular covariance function determine the extent to which these properties manifest themselves in the process. To illustrate this, consider again the noisy version of the squared exponential covariance function in Equation 2.4. In two spatial and one temporal dimensions, this covariance function has four free variables: the signal variance $\sigma_f$, the noise variance $\sigma_n$, the length-scale $l_s$ and the time-scale $l_t$. These variables are usually referred to as *hyperparameters*, since they determine the distribution of weights of an underlying parametric model (Rasmussen & Williams 2006, Section 2.1). This parametric model constitutes the bottom level of the model selection problem, and is fitted to the data with Equations 2.1 and 2.2. In other words, we need only concern ourselves with the upper two levels of the selection problem, since the GP equations take care of the bottom level.

Now, in practise, the type of covariance function is usually chosen by an expert based on the most distinctive properties of the process. For example, we might know that the squared exponential covariance function is an appropriate choice for modelling a specific environmental phenomena, but not the values of the hyperparameters. In general these are not known *a priori*, but have to be inferred from a set of observations. A standard way of doing this is the marginal likelihood (ML) method (Rasmussen & Williams 2006, Section 5.4). However, ML is known to be very sensitive to initial estimates of the hyperparameters, and often ends up in local maxima, resulting in very poor predictions. As an alternative, *Bayesian Monte Carlo* (BMC) (Rasmussen & Ghahramani 2003,

[5] For a $n \times n$ matrix $K(X, X)$, the Cholesky decomposition in line 3 takes $n^3/6$ operations, while solving the linear systems in steps 4 and 6 with triangular matrix $L$ takes $n^2/2$ operations.

Osborne et al. 2008), a principled Bayesian approach, was proposed that does not suffer from these problems.[6]

**Gaussian Processes in Related Work** Due to its versatility, the use of GPs for monitoring environmental phenomena is reasonably widespread. Low et al. (2008) use a GP to model environmental phenomena in a setting with mobile agents. Guestrin et al. (2005) and Krause & Guestrin (2005) use the GP in a similar fashion, with the aim of calculating highly informative placements (i.e. those that yield high observation value) for fixed wireless sensors. In order to do this, an initial deployment of a large number of sensors is required to collect a dataset of samples, from which the covariance function can be inferred. As an extension to this, Krause et al. (2006) also model the quality of the communication links between the sensors with a GP, to ensure a good trade-off between the informativeness of the sensor placement, and the expected communication cost between the sensors in the proposed deployment. However, both of these algorithms learn these models offline, and assume a dataset of samples is available prior to the deployment of the sensor nodes. In uncertain scenarios where time is of the essence, though, this assumption is unrealistic. For instance, in our intended application scenarios, agents are unlikely to have knowledge of the characteristics of their environment prior to their deployment.

So, in more time constrained and uncertain scenarios, such as those that are found in our intended application domains, the covariance function cannot be assumed to be known in advance. More specifically, for a solution to be *adaptive* (one of the requirements established in Chapter 1), sensors need to be able to learn the features of their surrounding environment *during deployment*. In GP terms, this implies that sensors need to be able to learn the covariance function online. To address this issue, Krause & Guestrin (2007) and Singh et al. (2007) present an exploration-exploitation approach that is capable of adapting the representation of the environment online, while moving a mobile agent towards more informative locations. In a similar vein, Osborne et al. (2008) and Garnett et al. (2009) perform online information processing on data streams from weather sensors using the BMC techniques mentioned earlier. With these techniques, it is possible to detect faulty sensors and predict missing sensor data using readings obtained from the other (functional) sensors. This is especially attractive from the perspective of *robustness*. As outlined in the introduction, robustness means that the operation of the mobile sensor network should not be interrupted when a single sensor fails. This technique allows the performance of the agents to degrade gracefully, since the failure of a single agent increases prediction error, but not catastrophically so.

[6] A full treatment of BMC is beyond the scope of this thesis. However, in an initial study we performed into agent-based information gathering for monitoring environmental phenomena (Stranders et al. 2008), we demonstrated the effectiveness of BMC in conjunction with a greedy decentralised coordination algorithm. Based on this study, in the remainder of this thesis we assume that the values of the hyperparameters are known, with the knowledge that BMC can be used when they are not.

In summary, in this section we have argued that the GP is a flexible and powerful method for performing regression over non-linear data, and, as such, is very suitable for modelling environmental phenomena. Given this, we choose GPs with covariance functions of the form in Equation 2.5 to represent and model environmental phenomena in Chapters 6 and 7, where we study several scenarios in which mobile agents monitor environmental phenomena.

#### 2.3.1.3 Other Approaches for Modelling Environmental Phenomena

Besides the linear regression and Gaussian processes, other techniques for representing environmental phenomena have been adopted in previous work. In what follows, we will briefly discuss these.

As opposed to performing explicit regression, Rahimi et al. (2004) recursively subdivide the environment into strata, ensuring that the variance of the observations made within each stratum is below a certain threshold. Each time the variance is found to exceed this threshold, a stratum is divided into four substrata, and the same sampling method is recursively applied on each of them. As a result, the spatial resolution is fine grained in areas where the environmental phenomenon is volatile and coarse grained in areas where it is not. The tree structure of strata, in combination with the mean and variance of each of the strata, constitutes the reconstruction of the measured phenomenon. While this is an elegant and simple solution, it does not recover the spatial and temporal correlations that are present within the environment. As such, although it presents an accurate spatial snapshot of the environment, it is not capable of extrapolating this snapshot in time, resulting in temporally deficient situational awareness.

In contrast, the Kalman filter is a common technique (Julier & Uhlmann 1997, Durrant-Whyte et al. 1990, Deshpande, Guestrin, Madden, Hellerstein & Hong 2005, Dash et al. 2005, Grocholsky et al. 2006) that does explicitly take into account the temporal dynamics of a phenomenon. Thus far, we have considered regression techniques for modelling phenomena whose spatio-temporal correlations are unknown and have to be identified. As opposed to these regression techniques, it is possible to explicitly take into account the system dynamics using the Kalman filter. For example, in target tracking it is known that the motion of a target is subject to the laws of Newtonian mechanics. Thus, based on a target's previous location, velocity and acceleration, it is possible to predict its current state (within a certain margin of error). The Kalman filter combines this prediction with noisy range and bearing observations (for example from a radar) to refine the target's state estimation.

Another attractive property of the Kalman filter is that it enables sensors to exchange and fuse beliefs about the state of their environment in a very communication efficient

manner; sensors simply need to exchange their current state vectors, which are subsequently fused with other sensors' states to increase tracking accuracy. In contrast, a drawback of the standard GP formulation is that, unlike the Kalman filter, belief sharing involves exchange of (possibly many) observations.

In the environments that we aim to deploy our approach in, however, sensors are dealing with a possibly wide variety of environmental phenomena for which the temporal dynamics are either unknown, or highly complex. For this reason, the Kalman filter is less suitable for modelling these phenomena, since it requires an explicit model of these dynamics. Instead, the GP is capable of recovering and approximating the intricate and complex dependencies between measurements in time and space with limited prior knowledge of the system dynamics.

In light of the above, a GP/Kalman filter hybrid, which combines the versatility of the GP and the communication efficiency of the Kalman filter would be desirable. Indeed, Reece & Roberts (2008) propose exactly such an approach, by showing the equivalence of the Kalman filter and GP. However, since the GP is more mature, and our work does not primarily focus on regression techniques, we prefer to use the GP. Further developments in this line of research should nevertheless therefore be monitored in order to determine whether it can form a basis for a more flexible method of processing information.

### 2.3.2 Wide Area Surveillance

Representing the environment in the wide area surveillance domain—the second exemplar application domain in Section 2.1—is significantly less complex than representation environmental phenomena. Representations found in the literature often keep track of the position of events or targets (Fitzpatrick & Meertens 2003, Vargas et al. 2003), or assume prior knowledge of the probability distribution for their arrival times and spatial distribution (Farinelli, Rogers & Jennings 2008, Giusti et al. 2007).

Given the simplicity of these representations, it is not surprising that the challenge of wide area surveillance lies not in representation, but rather in coordinating the agents' actions so as to maximise the number of events that are detected (or correctly classified). In Section 2.5 we will study previous work in this area, and look at these challenges in more detail.

### 2.3.3 Search and Patrol

The key challenge in search and patrol is creating a model of the attackers' behaviour and keeping track of its location. This behaviour might be strategic (in a game-theoretic sense), whereby the behaviour of the attackers and the pursuing agents are interdependent and both groups are aware of their conflicting interests (Agmon, Kraus & Kaminka

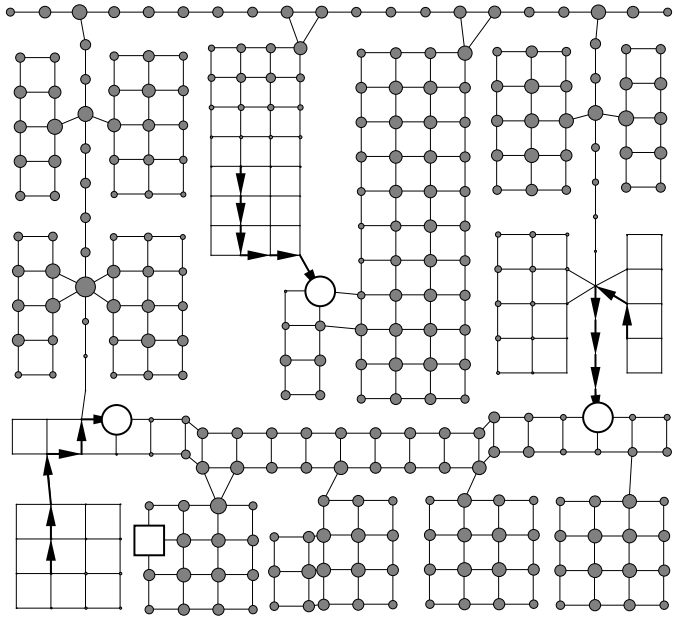


FIGURE 2.9: An example of a probabilistic map for pursuit evasion with three agents and an evader. The size of the grey circles is proportional to the estimated probability that the evader is there.

2008, Agmon, Sadov, Kaminka & Kraus 2008, Paruchuri et al. 2007), or non-strategic, whereby this behaviour is independent of the long-term behaviour of the pursuing agents (Hespanha et al. 1999, Vidal et al. 2001). Consequently, these different types of behaviour call for radically different techniques. Although this thesis is mainly concerned with the non-strategic setting, we will briefly discuss the strategic setting, as the applicability of our work to this setting is an open question, and is part of future work (see Section 8.2).

To model interactions with strategic attackers, various tool from game theory are used, depending on the type of setting. On the one hand, a Stackelberg game is appropriate when attackers are capable of observing the agent's strategy (Agmon, Kraus & Kaminka 2008, Agmon, Sadov, Kaminka & Kraus 2008, Paruchuri et al. 2007). A Stackelberg game models the so-called leader and follower roles of the agents (the leaders, who choose their actions first), and the attackers (the followers, who choose their action after having observed the agents' actions). If, on the other hand, agents and attackers choose their actions simultaneously, their interactions are commonly modelled with an *extensive-form game*, that represents their action sequence as a tree (Parsons 1978, Halvorson et al. 2009).

In the non-strategic setting, the key challenge is to maintain a probability distribution over the evader's location, known as a probabilistic map (Hespanha et al. 1999) (see Figure 2.9). In this case, a probabilistic map $p_e(e_t = v \mid Y_{t-1})$ estimates the probability that the evader is at location $v$, given previously made observations $\mathbf{O}_{\mathbf{A}}^{t-1}$. Updating this map with new observations to obtain $p_e(e_{t+1} = v \mid \mathbf{O}_{\mathbf{A}}^{t})$ proceeds in two steps:

1. Fuse new observations $O^t_{\mathbf{A}}$ with $p_e(e_t = v \mid \mathbf{O}^{t-1}_{\mathbf{A}})$ to obtain:

$$p_e(e_t = v \mid \mathbf{O}^t_{\mathbf{A}}) = \alpha p_e(e_t = v \mid Y_{t-1}) p(y_t \mid e_t = v, \mathbf{O}^{t-1}_{\mathbf{A}})$$

   Here, $\alpha$ is a normalising constant, and $p(O^t_{\mathbf{A}} \mid e_t = v, \mathbf{O}^{t-1}_{\mathbf{A}})$ depends on the sensing model, for example on the noise level of the agents' sensors.

2. Predict the motion of the evader given the evader's motion model $M$ (e.g. random):

$$p_e(e_{t+1} = v \mid \mathbf{O}^t_{\mathbf{A}}) = \sum_{v' \in V} p_m(e_{t+1} = v \mid e_t = v', M) p_e(e_t = v' \mid \mathbf{O}^t_{\mathbf{A}}) \tag{2.6}$$

Clearly, the last step is only possible if the evader's motion model is known. In Chapter 6, where we develop a decentralised algorithm for mobile agents, we return to the search and patrol domain, give more details on the two steps, and extend this model settings where the evader's motion model is unknown.

## 2.4 Valuing Observations

The second component of the centralised and agent-based architectures of information gathering systems (Figures 2.4 and 2.5) is responsible for valuing (future) observations in terms of their capability to improve situational awareness.

Now, the techniques discussed in the previous section provide a high-level representation of the raw observations, which enables information gathering agents to reconstruct the dynamics of their environment. The quality of this reconstruction critically depends on the observations that have been taken. To illustrate this with an example, consider a single agent with a limited battery life. If this agent samples from its environment at its maximum rate, it may well deplete its battery early in the day. Should we wish to reconstruct an environmental phenomenon during that day, the observations made by this agent are clearly less valuable than those obtained if it spaced out the same number of observations over the entire day. Put differently, the observations that the agent has collected are of little *value* in terms of situational awareness. In light of this, the concept of *observation value* is of critical importance, and is defined as follows:

**Definition 2.3** (Observation value)**.** The value of an observation is equal to the increase in the quality of situational awareness it contributes to.

Recall from Chapter 1 that quality of situational awareness does not only pertain to the prediction accuracy of the phenomena that exist within the environment, but also (and more importantly) to the extent that the impact of events and decisions on the mission objectives are understood. Therefore, the value of observations depends on the

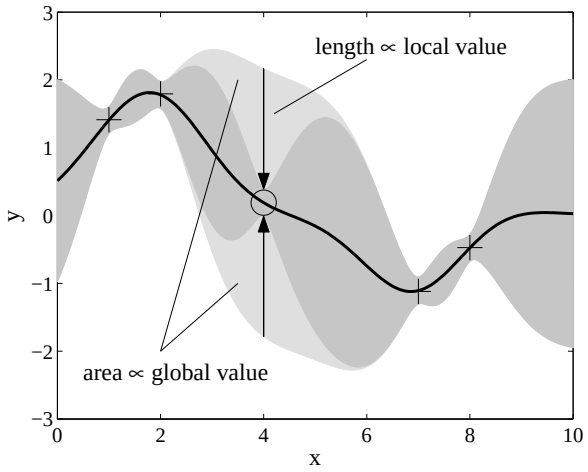



FIGURE 2.10: A GP illustrating the difference between the local and global value of a new observation. The local metric only measures the uncertainty reduction at the location where the observation (the circle) is made, whereas the global metric measures the uncertainty reduction across the entire space.

difference between the agents' belief and the ground truth, as well as the reduction of loss they achieve.

Now, when the value of future observations is known, an adaptive sampling algorithm can accurately decide where and when to sample in order to maximise the quality of situational awareness. In making this choice, it has to make a trade-off between bandwidth usage, movement and sampling (e.g. excessive communication and movement will leave little battery power for sampling, and *vice versa*), as well as the time it takes to reposition and make an observation. However, the only way the true value of a future observation can be known, is to actually make it and determine to what extent it improves situational awareness. Not surprisingly, this paradox presents a challenge. To address this challenge, the information processing techniques that were discussed in the previous section are commonly used to obtain an approximation of the value of an observation.

Generally speaking, in the state of the art, it is common to express this value in terms of a mathematical formalisation of the notion of expected "surprise" of a newly made observation. Equivalently, surprise can be thought of as the inverse of the confidence in the current model; the less confident an adaptive sampling algorithm is about (parts of) the current model of the environment, the greater the potential surprise (or value) of a new observation.

Observation value metrics can be divided into in two categories: *local* metrics, and *global* metrics. The former take into account the reduction in uncertainty (or conversely, increase in confidence) at the location where the observation is made. Figure 2.10 shows an example of this in the context of a Gaussian process. By making a new observation (indicated by a circle), the confidence bands around it tighten. The local value of this

observation is a function of the contraction (indicated by the arrows) of the confidence bands at the coordinate where it was made. In the literature, various metrics are used to measure local value, such as the width of the confidence bands (Kho et al. 2009), the variance (Osborne et al. 2008), Kullback-Leibler divergence (Padhy et al. 2010), the number of events detected (Farinelli, Rogers & Jennings 2008), and entropy (Ko et al. 1995). The last metric, entropy, is a concept from information theory and is often used in conjunction with GPs. Informally, entropy is a measure of the peakedness of the probability distribution of a random variable. The more peaked the distribution, the more confidence exists about the value of the variable, and the smaller the probability that we will be surprised by a new measurement. Formally, if we denote a set of observations as a $D$ dimensional vector of random variables $\mathbf{y}_*$ with a multivariate normal distribution (as is the case in the GP), the entropy $H$ is a function of its covariance matrix $\Sigma$ (Equation 2.2) only:

$$H(\mathbf{y}_*) = \frac{1}{2} D \ln(2\,\pi\,e) + \frac{1}{2} \ln(|\Sigma|) \tag{2.7}$$

Entropy as a value metric can also be used within the pursuit-evasion domain, where the random variables have a Bernouilli distribution (i.e. either the evader is present at a location, or it is not). Making an observation at location $v \in V$, the realisation (i.e. positive or negative) of which is modelled by Bernoilli random variable $\mathcal{X}_v$ is worth:

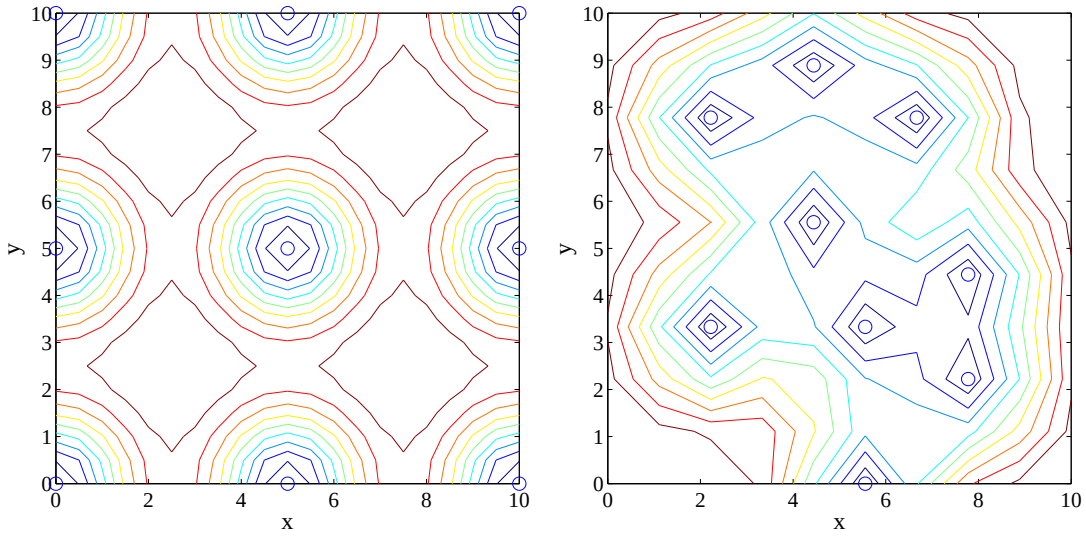
$$H(\mathcal{X}_v) = -p \ln(p) - (1-p) \ln(1-p)$$

where $p$ is the probability of the evader being at $v$.

Global metrics, in contrast to local metrics, measure the increase in situational awareness that results from a new observation over the *entire* environment. To again use the example in Figure 2.10, the global value of the new observation equals the contraction of the confidence bands over the entire space (indicated by the light grey areas). A commonly used global metric within information gathering is mutual information (MI) (Guestrin et al. 2005), which measures the *reduction* of entropy at all locations $V$ within the environment by making observations at locations $L$:

$$MI(V \setminus L; L) = H(V \setminus L) - H(V \setminus L \mid L) \tag{2.8}$$

Two considerations should be made for choosing between local and global metrics. Firstly, with reference to solution quality, the use of global metrics generally lead to better performance, since they more accurately represent the true impact of a new observation on the accuracy of the model. This effect was studied by Guestrin et al. (2005), who compare mutual information and entropy as a metrics for deploying fixed sensors in an environment modelled by a GP. By iteratively placing sensors at the location with maximal entropy, they found that a large proportion of the sensors end up along




(a) Entropy (b) MI

FIGURE 2.11: Placement of fixed sensors using the entropy and MI value metrics. Contours lines connect points of equal predictive variance (the lower the predictive variance, the higher the accuracy of the predictions).

the border of the environment, where they are maximally uncertain about each others' measurements, but where they also waste a large part of their range (Figure 2.11(a)). Using MI, however, the sensors were placed more centrally, and, as a result, use their resources more effectively (Figure 2.11(b)). Thus, their study confirms the superiority of the global mutual information metric over the local entropy metric.

The second consideration is computational overhead. All else being equal, the use of a global metric requires more computation than a local one. The reason for this is that the former needs to compute the effect of a new observation on the entire environment, instead of a single point. To illustrate this using the entropy and MI metrics, note that to evaluate MI in Equation 2.8, the entropy at all locations $V \setminus L$ has to be computed,[7] while to evaluate entropy, only a single point needs to be taken into consideration.

The choice between a local or global metric should be guided by the specific application domain and the representation of the environment that is most suitable for that domain. These representations vary significantly in complexity (as we have seen in Section 2.3), with a corresponding variation in the computational overhead required to evaluate a value metric. When the inherent computational overhead of a representation is low, a global metric should be naturally preferred. For example, as a result of the complex calculations involved in computing the variance (Equation 2.2), evaluating MI in a Gaussian process is more expensive than in pursuit-evasion where a probability map is used. In the latter case, MI is clearly more preferred than in the former case. We will

[7] That is, if we ignore the fact that a new observation generally only affects its neighbourhood, which is a small subset $V' \subset V$. However, even if this property is exploited by computing the entropy reduction at $V'$ instead of $V$, MI still requires more computation than entropy.

come back to the trade-off between computation and solution quality in Section 6.3.1.1, where we benchmark the entropy metric against the MI metric, and demonstrate that the latter indeed leads to better performance, but also to an increase in computation of two orders of magnitude.

## 2.5 Coordination

Now that we have studied methods for representing the environment and metrics for valuing observations, we can now address the third and final component of the architectures in Figures 2.4 and 2.5—coordination, which is the central focus of this thesis. Here, the model of the environment constructed by the first component (representation) and the value metrics of the second component (valuing observations) are exploited to determine where and when the agents should take observations.

In the state of the art, two different classes of algorithms can be distinguished: *offline* and *online* coordination algorithms. The former class of algorithms pre-compute a coordinated plan before the agents' deployment, instructing them when and where to observe their environment. The latter class of algorithms coordinate the the agents' actions during their operation.

Clearly, the latter class is more suitable when no prior knowledge of the environment is available, such as physical layout, or, for example, the spatio-temporal correlations of an environmental phenomenon. It is also more suitable in scenarios where agents need to be robust against *a priori* unknown events, such as the failure of one or more agents, or where they have to be able to adapt to a structural change to the way in which the environment behaves (for example, after the outbreak of a fire). However, when the environment can be considered static, sufficient knowledge about its layout is available, and precomputed plans inherently degrade gracefully, offline coordination algorithms might be preferred over online coordination algorithm because of their ease of implementation.

In this section, we discuss both classes in more detail from the perspective of a single agent and highlight their strengths and weaknesses in terms of the requirements defined in the introduction to this thesis. Then, in the next section, we provide an in-depth discussion of coordinating the actions of multiple agents, which is a key challenge in our work.

### 2.5.1 Offline Coordination Algorithms

A large body of previous work in the area of offline coordination algorithms is based on the notion of *submodularity*, which intuitively captures the diminishing returns of

making new observations: a new observation is more valuable if the agents have only made a few prior observations, than if they made many. More formally, let $f$ be a set function that computes the value of a set of observations. Function $f$ is submodular if it satisfies the following definition:

**Definition 2.4** (Submodularity)**.** A set function $f : 2^E \rightarrow \mathbb{R}$ defined over a finite set $E$ is called *submodular* if for $A \subseteq B \subseteq E$ and $e \in E$, $f(A+e) - f(A) \geq f(B+e) - f(B)$.

Under the assumption of submodularity of the value of observations, the *incremental value* $f(o+O') - f(O')$ of adding an observation $o$ to set $O'$ is larger than the incremental value $f(o+O) - f(O)$ of adding $o$ to a set $O \supset O'$. Throughout the rest of this thesis, we will often use this concept of incremental value, which is formally defined as:

**Definition 2.5** (Incremental Value)**.** The additional value (in terms of a set function $f$) obtained by adding a set $A \subseteq E$ to another set $B \subseteq E$ is called the *incremental value* of $A$, and is denoted as $\rho_A(B) = f(A \cup B) - f(B)$.

Thus, using this concept, the main property of a submodular function in Definition 2.4 can be restated as:

$$\rho_{\{e\}}(A) \geq \rho_{\{e\}}(B)$$

The importance of submodularity stems from the fact that many of the observation value functions we discussed in Section 2.4 (and in information gathering in general) are submodular, for example entropy, mutual information,[8] area coverage and number of events detected (in wide area surveillance, for example).[9] As a result, maximising observation value is equivalent to maximising a submodular function—a problem that has been extensively studied under cardinality constraints (Nemhauser & Wolsey 1978) and matroid constraints (Nemhauser et al. 1978). The former involves selecting $k$ elements from a set $E$, such that their value is maximised, i.e., finding $I^*$ such that $I^* = \arg\max_{I \subseteq E, |I|=k} f(I)$. Since this is a known NP-hard problem (Nemhauser & Wolsey 1978) no polynomial time algorithm for computing $I^*$ exists.

The problem of selecting the $k$ optimal elements from a set maps directly onto the problem of selecting $k$ observations, or, equivalently, deploying $k$ agents, in the presence of a submodular observation value function. This problem was studied by Guestrin et al. in a seminal paper (Guestrin et al. 2005), where they resort to approximation in order to avoid solutions whose computational complexity grows exponentially with the number of agents (thus violating our requirement of *scalability*). Specifically, they use an efficient (polynomial) greedy algorithm for maximising observation value. In each iteration, this algorithm adds the element $e$ with the highest incremental value to the

[8] Provided that $|V| > \frac{1}{2}|L|$ in Equation 2.8 (Guestrin et al. 2005).

[9] Additionally, these metrics can be weighted to reflect the importance of certain events or specific areas within the environment.

previously selected elements $I_G \in E$:

$$e = \arg\max_{e \in E \setminus I_G} \rho_{\{e\}} I_G$$

until the resulting set $I_G$ has the desired cardinality $k$. Interestingly, submodularity ensures that this algorithm has strong performance guarantees, as proved by Nemhauser & Wolsey (1978):

**Theorem 2.6.** *If $f$ is a submodular set function, then, for a given $k$, the approximation bound of the greedy algorithm $\frac{f(I_G)}{f(I^*)}$ is at least $1 - \left(\frac{k-1}{k}\right)^k$.*

For $k \to \infty$, this bound approaches $(1 - \frac{1}{e})$.[10] Thus, the greedy algorithm produces results that are at least $\approx 63\%$ as valuable as the optimal solution.

Guestrin et al. use this greedy algorithm to deploy fixed sensors in a GP with a known covariance function and hyperparameters. In addition to maximising observation value, Krause et al. (2006) simultaneously attempts to minimise communication cost between the deployed nodes. In order to do this, their algorithm exploits the *locality* property of the environment. This means that the correlation between observations taken at two distant locations is small enough to assume they are independent.[11] This property makes it possible to partition the environment in subspaces, such that observations made in different subspaces are independent. The problem is then solved by generating high quality deployments for each subspace with the greedy algorithm, which are subsequently connected to a communication network.

Singh et al. (2007) and Meliou et al. (2007) both extend the work of Guestrin et al. by exploiting the properties of submodularity and locality for pre-planning valuable paths for mobile agents. A decomposition strategy similar to the one in the sensor placement algorithm is used, whereby the environment is divided into grid cells that can be considered independent under the locality assumption. Their algorithms then perform a search over paths by connecting grid cells with high value, and return the agents to their starting location. Similarly, the approach by Zhang & Sukhatme (2007), which also uses mobile agents, performs a breadth-first search through the space of all feasible paths. A path is feasible if the boat has sufficient energy to follow it, and does not return to previously visited locations (recall from Section 2.2 that this is not a good strategy when the environment is constantly changing). The path that maximises the cumulative observation value obtained at the locations visited along that path is returned. Despite the use of heuristics, however, the use of breath-first search causes this algorithm to *scale* poorly with the size of the environment.

[10] Here, $e$ is the base of the natural logarithm, and not an element $e \in E$.

[11] For example, with the squared exponential covariance function in Section 2.3.1.2, the covariance between two observations drops exponentially with the distance between them. Thus, if two observations are sufficiently far apart, their covariance approaches 0, and they can be considered to be independent.

Within the pursuit and patrol domain, where the agents' main challenge is to detect and capture strategic attackers (Section 2.3.3) in an effort to minimise loss, we also find a number of offline, centralised algorithms (Paruchuri et al. 2007, Basilico et al. 2009, Agmon, Kraus & Kaminka 2008). These algorithms compute the optimal patrolling policy by formulating the problem as a mixed integer programming or linear programming problem. The optimal policy that is produced by these algorithms revisits all locations often (with varying frequencies based on their vulnerability or value), since attackers can appear anywhere at any time. However, since finding the optimal policy is an NP-hard problem (Basilico et al. 2009), these algorithms are only capable of solving small problem instances, and, as a result, are not *scalable*.

Now, from the point of view of the requirements laid down in Chapter 1, specifically those of *adaptiveness* and *autonomy*, offline algorithms are less suitable, since the pre-planning of paths does not allow agents to adjust these paths after the occurrence of *a priori* unknown events (e.g. the failure of an agent). Moreover, these algorithms compute these paths in a centralised way, and distribute the solution over the agents. This increases their vulnerability to failure of the central controller, which decreases their *robustness*.

Despite these drawbacks however, several elements of the techniques described above are of key importance to our work, all of which are related to the property of submodularity. First of all, the observation value function $f$ that we define in Chapter 3 is assumed to be submodular, since many observation gathering domains exhibit this property (including the exemplar domains from Section 2.1). Second of all, function $f$ is assumed to exhibit *locality*, which enables us to decompose the problem into simpler subproblems in Chapters 6 and 7 (similar to Singh et al. (2007) and Krause et al. (2006)). Thirdly, we make extensive use of the greedy algorithm and its associated *performance guarantees* for deploying fixed agents (Chapter 4) and computing infinite length patrols for mobile agents (Chapter 7)

In the next section, we will examine online approaches, which are better suited for adapting to uncertain and dynamic environments.

### 2.5.2 Online Coordination Algorithms

In contrast to the offline algorithms discussed in the previous section, online algorithms do not plan the agents' actions before deployment. Instead, they are *adaptive*, and select observations based on observations made during their operation. As was mentioned earlier, but is worth re-emphasising, the main benefit of online algorithms is their potential for being *robust* and their ability to *adapt* to *a priori* unknown events (two of the design requirements of Section 1.1).

Within the class of online coordination algorithms, we find a variety of techniques, which, for the purpose of analysis, we divide into three subclasses. Ordered by increasing length of their planning horizon these are: greedy, receding horizon and non-myopic. Each of these will now be discussed in turn.

Greedy algorithms maximise observation value for the next action only. They are therefore *reactive*, since they respond to immediate reward signals from their environment, and do not plan ahead. For instance, a fixed information gathering agent can take a new observation when the observation value exceeds a pre-set threshold (Osborne et al. 2008, Padhy et al. 2010) Within the literature on mobile information gathering agents, *potential fields* is an often encountered greedy and reactive technique (Kerr & Spears 2005, Pereira et al. 2004). Originally proposed for motion planning of robots (Dunias 1996), potential fields are used to attract mobile agents to areas with high observation value, while repelling them from areas with low value and other agents. For example, inspired by gas and fluid models, Kerr & Spears (2005) use potential fields to maximise area coverage with a swarm of mobile robots. Each robot is modelled as a (gas) particle that bounces off walls and other robots, effectively mimicking the (on-average uniform) dispersal of gas particles in a volume of space. Consequently, this method is especially attractive when a large number of robots is available, and degrades gracefully with failing robots. However, with only a few robots in a large environment, where robot-robot and robot-wall collisions are less likely, good dispersal and coverage are not guaranteed, resulting in poor situational awareness. Pereira et al. (2004) also use potential fields for controlling mobile robots, which collect data from fixed sensors (that are unable to communicate themselves). Sensors emit a potential field that attracts the mobile robot with a force proportional to the amount of uncollected observations that it has made, which, in turn, increases proportional to the rate at which the environment changes around the sensors. The mobile robots are therefore attracted to sensors in areas that are subject to rapid change, with corresponding high observation value. Finally, the work of Grocholsky (2002) and Grocholsky et al. (2005, 2006) can also be regarded an application of (a form of) potential fields. They refer to this technique as *information surfing*, whereby information agents continuously update their speed and heading so as to move in the direction of the steepest information value gradient. Generally speaking, due to their very limited lookahead, greedy algorithms are likely to get stuck in local minima, resulting in poor performance.

Receding horizon planning algorithms, the second class of online coordination algorithms, attempt to maximise observation value over an interval that encompasses more than a single action, but is shorter than the remaining mission time. It recomputes when the partially computed plan has been fully executed, or when the state of the environment changes unexpectedly. In general, this requires more computation time than greedily maximising the immediate next reward, because the algorithm needs to consider sequences of actions, instead a single one. Applications of this technique include

the control of single agent whose goal is to minimise uncertainty about its pose as well as maximise observation value (Martinez-Cantin et al. 2007), and pursuit evasion, where agents are continuously heading for the most likely location of the evader (a technique called "*global* greedy" by Vidal et al. (2001)).

Non-myopic algorithms, the third class of algorithms, consider the entire (remaining) mission time of an agent. The solution produced by these algorithms are paths of finite length (in case of a finite mission time) or patrolling policies (in case the mission time is unknown or infinite). In the latter case, agents are assumed to patrol their environment infinitely and continuously. While this is clearly an unrealistic assumption, it can be used to approximate settings wherein an agent is able to extract energy from its environment (Farinelli, Rogers & Jennings 2008, Kho et al. 2009). Within this class of algorithms, two lines of research are of particular relevance to this thesis. It is important to note, though, that both assume the phenomena have spatial, but no temporal correlations, making them less suitable for highly dynamic environments (see the discussion at the start of Section 2.3). The first combines the use of Gaussian processes to represent a phenomenon with the use of Markov decision processes (MDPs) to compute non-myopic paths for multiple mobile agents with a limited energy supply in an *online* fashion (Low et al. 2008). However, whilst this non-myopic approach computes high quality solutions, it incurs significant computational cost (it is only empirically evaluated for systems containing only two sensors), and is a centralised solution. The second line of research extends the techniques used for deploying fixed agents under the assumption of submodularity (Guestrin et al. 2005, Krause & Guestrin 2005, Krause et al. 2006) that were discussed in the previous section and develops a non-myopic control algorithm with strong theoretical bounds (Krause & Guestrin 2007). More specifically, it computes a policy for a single mobile agent to learn the hyperparameters of the GP online, and update the plan accordingly. Singh et al. (2009), in turn, generalise this work beyond environmental phenomena. They develop a polynomial divide and conquer algorithm that partitions the environment into clusters, such that the value of observations taken in different clusters are independent (i.e. additive). The algorithm then proceeds to find valuable paths within each cluster by using the greedy algorithm for submodular functions discussed in the previous section. Finally, these paths are connected in a way that ensures the energy cost of traversing it is within budget, the mobile agent returns to its starting position, and the observation value is maximised. Singh et al. provide theoretical bounds for solution quality and computation overhead, and introduce the concept of *adaptivity gap* to analyse when the environment has changed sufficiently to warrant recomputation.

Now, there is no general rule that prescribes whether a greedy, receding horizon, or non-myopic approach should be preferred. Instead, there are a few considerations that should be made in choosing one of these. Similar to the choice between local and global value metrics (Section 2.4), a trade-off between the *quality* of situational awareness and

*scalability* (two of the design requirements defined in Section 1.1) has to be made. All things being equal, the following rough relation can be observed: the longer the planning horizon, the more computation has to be performed, and the better the solution quality will be. For example, greedy algorithms are known to get stuck in local maxima, and can perform very poorly as a result. At the other extreme, non-myopic algorithms avoid the problem of local maxima, but the high computational overhead that they are often associated with makes these algorithms less scalable than algorithms with shorter lookahead. Moreover, the increase in solution quality is not necessarily (or usually) proportional to the length of the lookahead. Instead, it is characterised by strongly diminishing returns. This is particularly true when the environment is highly dynamic. In this case, planning far ahead can be very wasteful since changes in the environment and endemic uncertainty might quickly invalidate the assumptions on which a plan is based.

We will discuss the consequences of this trade-off between computational overhead and solution quality in the summary of this chapter. Next, however, we will focus on decentralised coordination, a special subclass of the class of online algorithms. Given the importance of this topic to this thesis, it is dealt with in a dedicated section, which follows next.

## 2.6 Decentralised Coordination

So far, we have implicitly assumed the existence of a centralised controller that controls the information gathering agents. However, in many cases, a centralised view of the environment is not available, or the costs associated with obtaining such a view are prohibitive (see the discussion of *autonomy* in Section 1.1). In these situations, there is a necessity for *decentralised* coordination algorithms. Using these algorithms, control is distributed over multiple agents, that compute joint, coordinated actions which maximise the observation value received as a team. As a result, the team is *robust* against failures (since no central point of failure exists), each agent controls its own actions (and thus is *autonomous*), and in many cases, the amount of computation that an individual agent needs to perform scales with the number of its neighbours, not with the size of the team, thus ensuring the *scalability* of the solution. Clearly, all these properties are desirable in terms of the design requirements laid down in Section 1.1.

The literature distinguishes three levels of decentralised coordination (Grocholsky 2002). Ordered by a decreasing amount of shared belief, these are:

1. *Cooperation* or *negotiated coordination* — agents share a common world view, and negotiate about their actions.

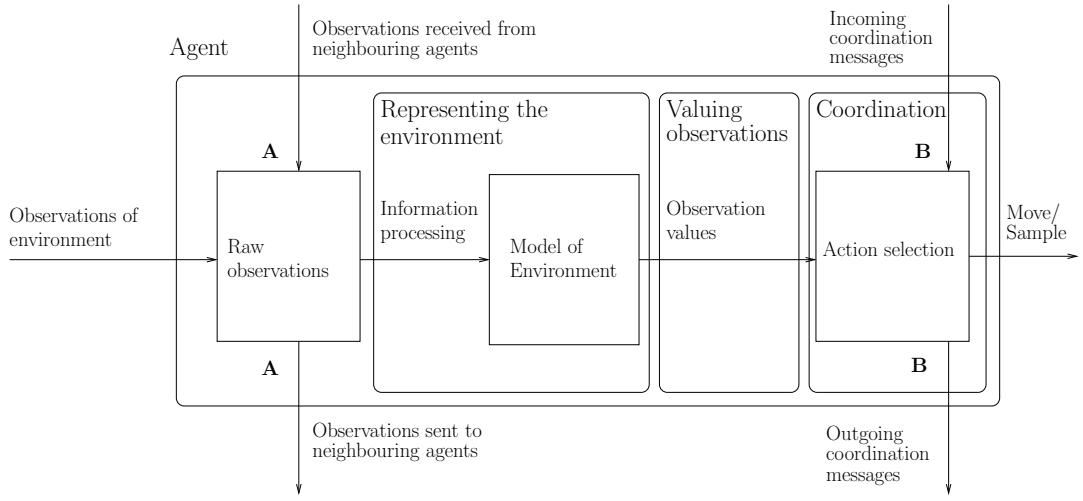



FIGURE 2.12: A general architecture of an information gathering agent.

2. *Un-negotiated coordination* — agents have a shared world view, but individually decide how to maximise observation value, without negotiating with their neighbours.

3. *Passive coordination* — agents do not communicate, but coordinate through passive coordination mechanisms, for example by observing each other's position or behaviour.

Returning to the architecture of an information gathering agent introduced at the start of this chapter, which is repeated in Figure 2.12, we can identify the impact of the three levels of coordination. Starting with the lowest level of coordination, *passive coordination* requires neither belief sharing, nor the exchange of negotiation messages. Consequently, arrows **A** and **B** are not present in the architecture of an agent operating in this mode. Moving up one level, *un-negotiated coordination* shares beliefs, but does not explicitly negotiate about actions. Thus, while arrows **A** are present for this mode of coordination, arrows **B** are not. Finally, in negotiated coordination mode, sensors both share beliefs, and negotiate about actions. As a result, both arrows **A** and **B** are present.

In the *negotiated cooperation* and the *un-negotiated coordination* modes, agents share an approximately common world view through the exchange of messages. These messages related to the observations that they have made, but do not necessarily contain only raw samples obtained from the environment. Rather, they can also contain a summary or aggregation of the samples made so far. In particular, it is very desirable to make the message size independent of the amount of samples its contents are based on. For example, the use of the information variant of the Kalman filter lends itself to efficient communication, since only the agent's current belief needs to be communicated with others (Grocholsky et al. 2006, Reece & Roberts 2005). To the best of our knowledge,

there does not exist any work that achieves a similar efficiency for the distributed use of the GP.[12]

The crucial difference between cooperation in negotiated or un-negotiated mode is that, in the latter mode, each agent unilaterally chooses the best (local) decision, without informing other agents of its intentions. As a result, agents will usually become aware of the impact of the collective decision after the fact, at which point it is too late to correct for any possible conflicts. In negotiated cooperation, however, these conflicts can be avoided because agents make their decisions after a negotiation phase with their neighbours, which is aimed at maximising the collective observation value received by the agents. Compared to un-negotiated coordination, negotiated coordination requires more computation and communication, but is also likely result in better performance.

Now, it is often the case that the observation value received by a team of agents can be *factorised* into a sum of observation values received by individual agents. This is due to the fact that many observation value functions exhibit the property of *locality* (see Section 2.4). Locality implies that observations taken sufficiently far apart in space are (almost) independent (i.e. their value is additive). When we substitute the term 'utility' for 'observation value' (two terms we use interchangeably in this thesis), the sum of observation values is commonly referred to as *social welfare* within the multi-agent systems literature.[13] Within this setting, we wish to find an action for each agent, such that the team utility (and thus observation value) is maximised. More formally, let $\mathbf{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_M\}$ denote the set of agents, and $\mathbf{p} = \{p_1, \ldots, p_M\}$ the set of discrete control or action variables with domain $\mathcal{D}_i = \{a_i^1, a_i^2, \ldots, a_i^{k_i}\}$. Furthermore, every agent $\mathcal{A}_i$ has a utility (or observation value) function $U_i(\mathbf{p}_i)$ that depends on the set $\mathbf{p}_i \subset \mathbf{p}$ of action variables belonging to agents whose observations are not independent with those of $\mathcal{A}_i$. The goal of the team is then to find the optimal action $\mathbf{p}^*$ that maximises the sum of the agents' utilities:

$$\mathbf{p}^* = \arg\max_{\mathbf{p}} \sum_{i=1}^{M} U_i(\mathbf{x}_i) \tag{2.9}$$

Furthermore, in order to enforce a decentralised solution, we assume that each agent only has knowledge of, and can directly communicate with a few neighbouring agents that influence its utility directly. As a result, the complexity of the computation an agent has to perform depends its number of neighbours, and not on the total number of agents in the team, making the decentralised solution scalable. Clearly, these properties are highly desirable in light of the requirements of scalability, robustness and autonomy (see Section 1.1).

[12] Although the methods proposed by Reece & Roberts (2008) seem very promising. As mentioned before, an investigation into the applicability of these methods is part of future work.

[13] The same problem is also referred to as the optimal control problem in control theory (Paskin et al. 2005).

Against this background, we can treat Equation 2.9 as a Distributed Constraint Optimisation Problem (DCOP) (Modi et al. 2003), in which "multiple cooperative agents control one or more variables and work together to optimise a set of constraints that exists upon these variables".[14] In recent years, this type of problem has been studied extensively, which has led to a wide range of algorithms that can be readily applied to solve them. Such algorithms can be broadly divided in two main classes: complete algorithms that generate optimal solutions such as ADOPT (Modi et al. 2005), OptAPO (Mailler & Lesser 2008), and DPOP (Petcu & Faltings 2005); and approximate algorithms such as the Distributed Stochastic Algorithm (DSA) (Fitzpatrick & Meertens 2003), Maximum Gain Message (Maheswaran et al. 2005), and k-optimal algorithms (Kiekintveld et al. 2010).

Now, while complete algorithms provide guarantees on the solution quality, they also exhibit an exponentially increasing coordination overhead (either through the size and/or number of messages exchanged, or in the computation required by each agent) as the number of agents in the network increases. Conversely, approximate algorithms require very little local computation or communication, but often converge to poor quality solutions because agents do not propagate information across the entire team. Rather, local information is only used in coordinating neighbouring agents. For example, using DSA each agent communicates its preferred action (e.g., the one that will maximise its own utility) based on the current preferred actions of its neighbours only.

However, there exists a class of algorithms usually referred to under the framework of the Generalised Distributive Law (Aji & McEliece 2000), that constitute a compromise between the extremes represented by these two classes, and can be used to obtain good approximate solutions. These algorithms have been widely used in the field of information theory and probabilistic inference to decompose complex computations on single processors (MacKay 2003), and more recently both complete and approximate algorithms from this framework have been applied to the coordination of networked sensing devices within the domain of discrete action parameters (Paskin et al. 2005, Farinelli, Rogers, Petcu & Jennings 2008, Kim et al. 2010, Waldock et al. 2008).

### 2.6.1 The Max-Sum Algorithm

In particular, one of the approximate algorithms, called the max-sum algorithm, has been shown to generate solutions closer to the optimum than previous approximate stochastic DCOP algorithms (Farinelli, Rogers, Petcu & Jennings 2008). It does so with an acceptable computation and communication overhead when benchmarked against representative complete algorithms (specifically DPOP), and it has been shown to be robust to message loss, making it very relevant in the context of the design requirements of this thesis.

[14] `http://teamcore.usc.edu/dcop/`

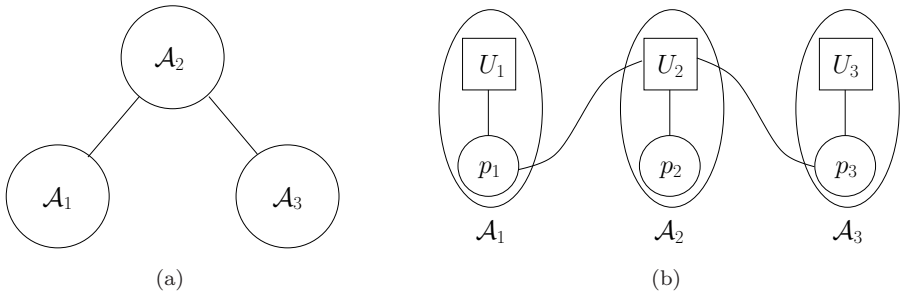



FIGURE 2.13: Diagram showing (a) the interactions of agents $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, (b) factor graph representing the dependencies between the agents actions.

In more detail, the max-sum algorithm operates on a *factor graph* that represents the optimisation problem described in Equation 2.9. A factor graph is an undirected bipartite graph in which vertices represent variables $p_i$ and utility functions $U_j(\mathbf{p}_j)$. In such factor graphs, an edge exists between a variable $p_i$ and a function $U_j$ iff $p_i \in \mathbf{p}_j$, (i.e., $p_i$ is a parameter of $U_j$). For example, Figure 2.13(a) shows three interacting agents, and 2.13(b) shows the factor graph encoding the interactions between them. In order to have a truly decentralised computation, the function that represents an agent's utility, as well as the variable that represents the agent's action variable, are assigned to the physical computational unit associated with that agent.

Using the max-sum algorithm, each agent $\mathcal{A}_i$ is able to compute the *marginal utility* of its action variable $p_i$:

$$\tilde{U}_i(p_i) = \max_{\mathbf{p_i} \setminus p_i} \sum_{i=1}^{M} U_i(\mathbf{p}_i) \tag{2.10}$$

in a distributed way (i.e. based on local information and communication with direct neighbours). Here, $\tilde{U}_i(p_i)$ is the *marginal function* of $p_i$, where for any state $a \in \mathcal{D}_i$, $\tilde{U}_i(a)$ is equal to the maximum value the global objective function (Equation 2.9) can attain if $p_i = a$. Thus, after computing the marginal function agent $\mathcal{A}_i$ can compute action $p_i^*$ that maximises global welfare, as follows:

$$p_i^* = \arg\max_{p_i} \tilde{U}_i(p_i) \tag{2.11}$$

This function $\tilde{U}_i(p_i)$ is computed by message passing between the functions $U_i$ and the variables in $\mathbf{p}_i$ as follows:

- **From variable $p_i$ to function $U_j$:**

$$q_{i \to j}(p_i) = \alpha_{ij} + \sum_{k \in \mathcal{M}_i \setminus j} r_{k \to i}(p_i) \tag{2.12}$$

where $\mathcal{M}_i$ is a set of function indexes, indicating which functions are adjacent to variable $p_i$ in the factor graph,[15] and $\alpha_{ij}$ is a normalising constant to prevent values from growing endlessly to the point of calculation error in cyclic factor graphs.

- **From function $U_j$ to variable $p_i$:**

$$r_{j\to i}(p_i) = \arg\max_{\mathbf{p}_j \setminus p_i} \left[ U_j(\mathbf{p}_j) + \sum_{k \in \mathcal{N}_j \setminus i} q_{k\to j}(p_k) \right] \tag{2.13}$$

where $\mathcal{N}_j$ is a set of variable indexes, indicating which variables are adjacent to function $U_j$ in the factor graph.

In acyclic factor graphs, the messages exchanged between variable $p_i$ and function $U_j$ are functions of a single variable ($p_i$) that represent the maximum aggregate utility possible over the respective halves of the graph formed by removing the edge between $p_i$ and $U_j$, for each of $p_i$'s possible states $a \in \mathcal{D}_i$.

At any time during the propagation of these messages, each agent $\mathcal{A}_i$ is able to compute the value of its action variable $p^*$ that maximises Equation 2.9. This is done by locally calculating the *marginal* function $\tilde{U}_i(p_i)$ with respect to variable $p_i$ from Equation 2.10, by summing the messages received by agent $\mathcal{A}_i$'s variable node:

$$\tilde{U}_i(p_i) = \sum_{j \in \mathcal{M}_i} r_{j\to i}(p_i) \tag{2.14}$$

and hence finding:

$$p_i^* = \arg\max_{p_i} \tilde{U}_i(p_i) \tag{2.15}$$

The earlier statement that Equation 2.15 results in the optimal joint action has to be somewhat qualified. In particular, this statement holds when the factor graph is acyclic (Aji & McEliece 2000).[16] In this case, the algorithm is guaranteed to converge, Equation 2.14 is the true marginal of $p_i$, and, as a result, Equation 2.15 is indeed the optimal solution. However, when applied to cyclic graphs, there is no guarantee of convergence, and the algorithm returns an approximation of the marginal function $\tilde{U}_i(p_i)$, which can theoretically result in arbitrarily bad solutions. Despite this, extensive empirical evidence has demonstrated that the family of algorithms to which max-sum belongs, generates good approximate solutions (Kschischang et al. 2001, Frey & Dueck 2007).

[15] For example, in Figure 2.13(b), $\mathcal{M}_1 = \{1\}$, $\mathcal{M}_2 = \{1, 2, 3\}$, and $\mathcal{M}_3 = \{3\}$.

[16] This implies that the problem can be broken into two simpler subproblems by removing a single dependency between two agents.

**The Bounded Max-Sum Algorithm** Despite the existence of this empirical evidence, however, the lack of performance guarantees limits the applicability of the max-sum algorithm in many application domains (particularly life-critical ones, such as disaster management and military surveillance), since the existence of pathological behaviour can not be ruled out. A solution to this problem was proposed by Farinelli et al. (2009), which involves the removal of dependencies between utility functions and variables, such that the resulting factor graph is acyclic. As a result, this variant of the max-sum algorithm, called the *bounded* max-sum algorithm, is guaranteed to converge, and provides a bounded approximation of the original problem, due to the removal of dependencies.

In more detail, the bounded max-sum algorithm attempts to remove the dependencies that have the least impact on the solution quality. The impact of a dependency $e_{ij}$ between $p_i$ and $U_j$ is referred to as its *weight* $w_{ij}$, and is defined as:

$$w_{ij} = \max_{\mathbf{p}_j \setminus p_i} \left[ \max_{p_i} U_j(\mathbf{p}_j) - \min_{p_i} U_j(\mathbf{p}_j) \right] \tag{2.16}$$

Here, weight $w_{ij}$ represents the maximum impact variable $p_i$ can have on the value of function $U_j$. As a result, if the dependency is ignored, the maximum difference between the computed solution and the optimal will be $w_{ij}$. Since this difference is to be as small as possible, the objective is to find a subtree of the original factor graph $FG$, such that the sum $W$ of weights of the removed dependencies is minimised:

$$W = \sum_{e_{ij} \in C} w_{ij}$$

where $C$ is the set of removed edges. This problem is equivalent to finding a maximum spanning tree of $FG$, which is computed in a decentralised fashion using the GHS algorithm (Gallager et al. 1983).

To obtain the approximate solution, the standard max-sum algorithm is run on this spanning tree, where utility functions $U_j$ that have had dependencies removed are evaluated by minimising over all values of the removed variables $\mathbf{p}_j^c$. Consequently, the algorithm produces an optimal solution to the following problem:

$$\tilde{\mathbf{p}} = \arg\max_{\mathbf{p}} \sum_i \min_{\mathbf{p}_j^c} U_j(\mathbf{p}_j) \tag{2.17}$$

Farinelli et al. (2009) prove that relation between the (unknown) optimal solution $V^* = \sum_j U_j(\mathbf{x}_j^*)$ and the approximate solution $\tilde{V} = \sum_j U_j(\tilde{\mathbf{x}}_j)$ for a particular factor graph $FG$ is given by:

$$V^* \leq \rho(FG)\tilde{V} \tag{2.18}$$

where $\rho(FG)$ is the data-dependent (i.e. dependent on the specific problem encoded by factor graph $FG$) approximation ratio. This ratio is computed as:

$$\rho(FG) = 1 + \frac{\tilde{V}^m + B - \tilde{V}}{\tilde{V}} \tag{2.19}$$

and $\tilde{V}^m = \sum_j \min_{\mathbf{p}_j^c} U_j(\tilde{\mathbf{x}}_j)$ represents the optimal solution to the tree structured constraint network.

**Message Passing Schedule** Now that we have defined the messages that are exchanged between the variables and functions, and discussed the bounded max-sum algorithm, one final issue needs to be addressed. This issue pertains to the order in which messages are sent, and the number of messages that need to be exchanged until the algorithm terminates.

In more detail, in a cyclic factor graph,[17] the messages exchanged between functions and variables described in Equations 2.13 and 2.12 may be randomly initialised, and then updated whenever an agent receives an updated message from a neighbour; there is no need for a strict ordering or synchronisation of the messages. In addition, the calculation of the marginal function in Equation 2.14 can be performed at any time (using the most recent messages received), and thus, agents have a continuously updated estimate of their optimum action.

The solution computed by the algorithm depends on the structure of the agents' utility functions, and, in general, three behaviours can be observed:

1. The preferred actions of all agents converge to fixed actions that represent either the optimal solution, or a solution close to the optimal, and the messages also converge (i.e. the updated message is equal to the previous message sent on that edge), and thus, the propagation of messages ceases.
2. The agents' preferred actions converge as above, but the messages continue to change slightly at each update, and thus continue to be propagated around the network.
3. Neither the agents' preferred actions, nor the messages converge and both display cyclic behaviour.

Thus, depending on the problem being addressed, and the convergence properties observed, the algorithm may be used with different termination rules:

[17] This message passing schedule applies to acyclic graphs as well, but more efficient methods exist for this type of factor graphs (Farinelli, Rogers, Petcu & Jennings 2008).

1. Continue to propagate messages until they converge, either changing the action of the agents continuously to match the optimum indicated, or only after convergence has occurred.

2. Propagate messages for a fixed number of iterations per agent (again either changing the action of the agent continuously or only at termination).

The first termination rule favours the quality of the solution. When the algorithm converges, it does not converge to a simple local maximum, but to a neighbourhood maximum that is guaranteed to be greater than all other maxima within a particular large region of the search space (Weiss & Freeman 2001). Depending on the structure of the factor graph, this neighbourhood can be exponentially large. However, only limited guarantees for convergence of the max-sum algorithm exist, and for general factor graphs the algorithm might not converge. For practical applications, therefore, the second termination rule is often preferred. In fact, empirical evidence shows that the max-sum algorithm reaches good approximate solutions in a number of iterations that is less than the diameter of the graph, to allow information to be exchanged between any pair of nodes in the graph (Farinelli, Rogers, Petcu & Jennings 2008). In addition, in dynamic scenarios where the utilities of the agents or the interactions between them change over time, the max-sum algorithm can run indefinitely without any termination rule; each agent can decide at every cycle which action to choose based on Equation 2.14, and operates on a continuously changing coordination problem. In light of this, in this thesis, we opt for the second termination rule.

This concludes our discussion of the max-sum algorithm. Due to its robustness to message loss, the fact that it is fully decentralised, and that it has an acceptable computation and communication overhead, the max-sum algorithm is an attractive option to serve as a basis for a coordination algorithm for information gathering agents. In Chapter 6, we will therefore show how max-sum can be applied to an information gathering domain with mobile agents, resulting in a decentralised coordination algorithm that has many of the required properties stated in Chapter 1. Furthermore, since many domains are characterised by continuous action variables (for instance, agents in the wide area surveillance setting in Figure 2.2, whose viewing angle can range between 0 and 360 degrees), instead of the discrete variables that the standard max-sum algorithm supports, we extend the max-sum algorithm to continuous variables and utility functions in Chapter 5.

## 2.7 Required Scale of a Team of Agents

In the previous sections we studied related work using the general architecture for an information gathering system. A few important questions remain, which relate to the

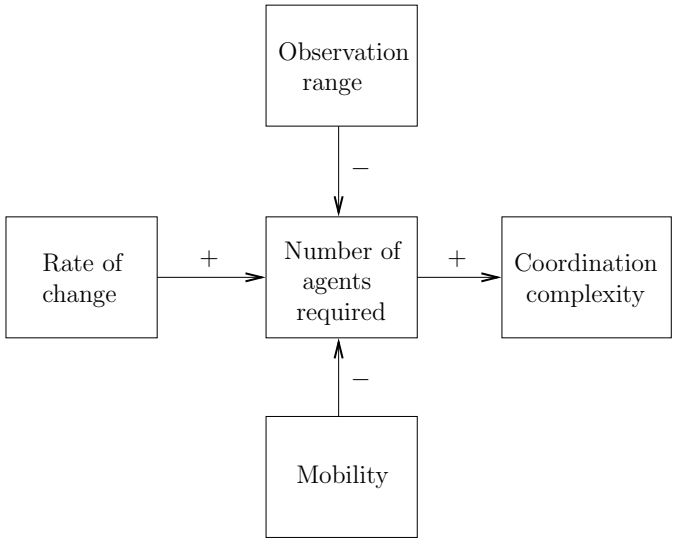



FIGURE 2.14: The effect of changing the observation range, rate of change and mobility on the scale of a team of agents required to achieve a fixed level of performance. The + and − indicate a positive and negative relation respectively.

required scale of a team of information gathering agents. More specifically, how does the rate of change of the environment, the mobility and the observation range of the agents impact on the number of agents required to achieve a certain level of performance? Fixing the desired level of performance (expressed in terms of observation value), Figure 2.14 summarises the relations between these variables and the number of agents that are needed to achieve this level of performance.

The first variable, rate of change, relates to the speed at which the environment changes in space, time, or both. The rate of change is roughly inversely proportional to the correlations that exists between observations made at different points in space-time. The weaker these correlations, the less information is revealed by making an observation. As a result, in order to maintain a constant level of performance, a larger number of agents is needed as the rate of change increases.

The second variable, observation range, models how much of its environment an agent can observe at any given time. This depends in part on the environment itself (e.g. on the number of obstructions present in it), the nature of the application domain (for example, radiation strength can only be measured at the agent's location, while intrusions can be detected within the agent's viewing area), and the type and quality of sensing available to the agents. Clearly, there exists a negative relation between the agents' observation range and the size of the team required to achieve a fixed level of performance.

The third variable is mobility, which relates to the speed of agents relative to the size of the environment. At one end of the scale, we find fixed agents, such as the ones embedded in fixed sensor networks, while at the other we find UGVs and UAVs. The

greater an agent's mobility, the larger the fraction of the environment it can observe, thus requiring fewer agents to achieve the same level of performance.

Now, in general, the complexity involved in coordinating a team of agents increases with its size. This is because more agents imply more interactions between the agents. The magnitude of this complexity increase depends on the nature of the problem and the type of coordination algorithm used. For instance, the complexity of solving a problem with dependencies between all pairs of agents with an optimal algorithm (such as DPOP) increases exponentially faster than the complexity of solving a problem with sparse interactions with an approximate algorithm (such as max-sum).

Of the relations mentioned above, one in particular—the (negative) relation between mobility and the number of required agents—is of specific importance to this thesis. In more detail, the decentralised coordination algorithms developed in Chapters 4 and 5, which are concerned with fixed agents, need to be able to coordinate a much larger number of agents to achieve the same level of performance as the algorithms for coordinating mobile agents which are discussed in Chapters 6 and 7. Thus, the algorithms for fixed agents need to be more *scalable* than those for their mobile counterparts. Fortunately, as we will see in these chapters, this need can be satisfied due to the fact that the action spaces of fixed agents are much less complex than those of mobile agents.

## 2.8 Summary

Let us recapitulate what we have discussed in this chapter and determine the impact of previous work on the work presented in this thesis. At the start of this chapter, we introduced three exemplar information gathering domains—monitoring environmental phenomena, wide area surveillance and search and patrol—that will be addressed in this thesis. While seemingly different, we showed that previous approaches for these domains exhibit several shared patterns and similarities by analysing them through the lens of a general architecture for an information gathering system (Figures 2.4 and 2.5). This architecture consists of three major components: a representation of the environment, a value metric for observations and a component responsible for coordinating the agents' actions in order to maximise observation value. In what follows, we briefly summarise our main findings for every component in turn.

In discussing related work from the perspective of the first component, representing the environment, we have seen that there exists a wide variety of techniques. Of all the three components of the architecture, these techniques are the most domain dependent since they are responsible for transforming observations of inherently domain dependent features into an abstract model of the environment. We discussed the most commonly used ones for each of the three exemplar domains. For monitoring environmental phenomena, we have seen that the GP is a very versatile and powerful tool for accurately

modelling the spatial and temporal dynamics. Specifically, from the perspective of our requirement of *quality*, insight into these dynamics for achieving situational awareness is important, and the GP allows us to recover them in a straightforward way. However, the versatility of the GP comes at a higher computational cost than, for example, linear regression. Nevertheless, we believe the GP is an attractive technique, since it does not put restrictions on the class of phenomena that can be modelled. Consequently, it is our preferred technique for monitoring environmental phenomena in Chapters 6 and 7.

For wide area surveillance, and search and patrol, the challenge of representing the environment is less complicated. In the former, this challenge is limited to maintaining a map of their coordinates, or learning their spatial distribution. In the latter, search and patrol, the belief over the possible locations of *non-strategic* attackers, which are the focus of this thesis, is represented as a probability map. We showed these probability maps are updated based on new observations and a behaviour model of the attackers.

In terms of valuing observations, the second component, we defined the value of observations in terms of their contribution towards improving the quality of situational awareness. We discussed the key mathematical concept of *submodularity*, which captures the diminishing returns of making additional observations, and showed that many observation value metrics found in many information gathering domains exhibit this property. Because of this, the observation value function in the central problem formulation in Chapter 3 is assumed to be submodular, and the greedy algorithm for maximising submodular functions forms the foundation of the algorithms we develop in Chapters 4 and 7. Furthermore, we grouped value metrics into local and global ones, and discussed the trade-off between solution quality and computation overhead that the choice between them entails. Generally speaking, the use of global metrics (which take into account the impact of making an observation on the entire environment) results in better solutions at the cost of a higher computational overhead, compared to the use of local metrics (which only take into consideration the reduction surprise at the location where an observation is taken). Therefore, when the computational overhead associated with the technique for representing the environment is high, it is often prudent to resort to a local metric. In Chapter 6, where we use the GP for representing environmental phenomena, we study this trade-off in more detail by comparing the local entropy metric against the global mutual information one.

We demonstrated that third and final component, which is responsible coordinating the agents' actions, can be studied in a domain independent fashion using our proposed architecture. We grouped existing coordination algorithms into an offline and an online subclass. Particularly. we discussed how online algorithms are more appropriate in dynamic and uncertain scenarios, since they are more *robust* to failure and are better able to *adapt* to *a priori* unknown events (two of the requirements defined in Chapter 1) than their offline counterparts. In light of this, the algorithms presented in this thesis are almost exclusively online, with the exception of the online/offline hybrid algorithm

| | **Chapter** | | | |
|---|---|---|---|---|
| **Property** | **4** | **5** | **6** | **7** |
| Lookahead | Greedy | Greedy | Receding horizon | Non-myopic |
| Value Metric | Global | Global | Both | Both |
| Online vs. Offline | Online | Online | Online | Offline/Online |
| Mobile vs. Fixed | Fixed | Fixed | Mobile | Mobile |
| # Agents | > Hundreds | Hundreds | Tens | Tens |

presented in Chapter 7. This algorithm computes a baseline offline solution, which is improved upon through online coordination.

We further subdivided the online coordination algorithms into three categories, based on the length of lookahead: greedy, receding horizon and non-myopic. We argued that deciding an appropriate lookahead involves a trade-off between *quality* and *scalability*: in general, increasing the lookahead produces better solutions (although subject to diminishing returns), but also results in a dramatic increase of computation (which is often exponential in the length of the action sequence). Moreover, the non-myopic algorithms we discussed in Section 2.5.2 are of particular interest since they are capable of give *performance guarantees*.

In light of this, in this thesis, we propose algorithms with varying degrees of lookahead. On the one hand, the algorithms for deploying and coordinating fixed agents presented in Chapters 4 and 5 can be regarded as greedy: the former uses an algorithm similar to the greedy algorithm for maximising submodular functions discussed in 2.5; the latter uses the max-sum algorithm to find the immediate next joint (i.e. coordinated) action that maximises observation value. On the other hand, the decentralised coordination algorithms for controlling mobile agents in Chapters 6 and 7 require are receding horizon (with an adjustable lookahead) and non-myopic control (with performance guarantees) algorithms respectively.

As a special subclass of online algorithms, we discussed decentralised coordination. In particular, we focused on the max-sum message-passing algorithm for decentralised coordination and argued that it exhibits desirable properties as robustness, scalability and autonomy. In light of this, in this thesis, we use the max-sum algorithm for coordinating both fixed (Chapter 5) and mobile agents (Chapters 6 and 7) and develop several extensions to it. In particular, in Chapter 5, we extend max-sum to problems with continuous action variables, and present various techniques for improving its computational efficiency in Chapter 6.

Finally, we discussed the required scale of a system of information gathering agents, based on several intrinsic properties of the application domain. These properties—mobility, rate of change and observation range—impact in different ways on the level of achieved performance. One relation in particular, the (negative) relation between

mobility and the number of required agents, is specifically important to this thesis. In particular, in Chapters 4 and 5, which deal with fixed agents, a much larger number of agents is required to achieve the same level of performance as the mobile agents that are discussed in Chapters 6 and 7. As a result, the algorithms for fixed agents need to be (and, indeed, are) more scalable than those for their mobile counterparts.

Table 2.8 summarises the properties of the algorithms presented in this thesis.

As we have seen in this chapter, there is a wide variety of bespoke algorithms for different types of environments, solving a wide spectrum of seemingly different problems. However, by analysing these algorithms from the perspective of the architecture for information gathering systems, we have demonstrated that its three components can be described in an increasingly domain independent fashion. Thus, whilst the component responsible for representing the environment transforms domain dependent observations into an abstract model of the environment, and the observation value component depends on this model for ranking future observations, the coordination component is fully shielded from domain dependent features through the domain independent concept of *observation value*. In the next chapter, we exploit this property by formulating the problem of coordinating information gathering agents exclusively in terms of this concept. By so doing, the decentralised coordination algorithms that we develop in this thesis—which operate directly on this problem—are applicable to a wide spectrum of problem domains.

# Chapter 3

# The Multi-Agent Information Gathering Problem

In this chapter we present a general formalisation of the multi-agent information gathering problem. This formulation is domain independent, and therefore makes no reference to any domain specific properties (e.g. targets, environmental phenomena, or intruders). As we have seen in Chapter 2, this can be accomplished through the use of the concept of *observation value*, which abstracts from the chosen representation of the environment, and defines the value of observations in terms of their contribution to improving situational awareness.

Our formalisation is inspired by that of Singh et al. (2007), which we have extended with a temporal dimension (i.e. the property of *temporality*), and placed in an agent-based setting where each sensor is modelled as an autonomous agent that possesses limited and local knowledge.

In what follows, we describe first the features that are shared between all information gathering domains. Then, we describe additional features that apply to settings with mobile agents. Finally, we describe the agents' objective.

Now, the multi-agent information gathering problem is given by:

- A set of agents: $\mathbf{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_M\}$
- A set of spatial coordinates $L$ embedded in Euclidian space, in which the agents take measurements. The Euclidian distance between two coordinates $u, v \in L$ is denoted by $\mathrm{d}(u, v)$.
- A discrete set of temporal coordinates $T = \{1, 2, 3, \ldots\}$, that specify when the agents can take measurements.

- A set of spatio-temporal observation coordinates $O = L \times T$. An element $o \in O$ is called an observation. We will use the notation $o(l) \in L$, $o(t) \in T$ and $o(m)$ to refer to the $o$'s spatial coordinate, its temporal coordinate, and its realisation. The latter refers to what was actually measured, and can be a scalar (e.g. temperature) or binary (e.g. whether an intruder was detected).

- A set function $f : 2^O \to \mathbb{R}^+$ that assigns *observation value* to a set of observations. This value is proportional to the situational awareness this set of observations brings about. Thus, function $f$ should assign a value to a set of observations $O$, based on the prediction accuracy, as well as the loss prevented by making these observations. This function has the following (optional) properties:

  **Property 1** (Non-decreasing)**.** $f$ is non-decreasing: $\forall A \subseteq B \subseteq O$, $f(A) \leq f(B)$. Thus, acquiring more observations never 'hurts'.

  **Property 2** (Submodularity)**.** $f$ is submodular: $\forall A \subseteq B \subseteq O$ and $\forall o \notin A$:

  $$f(A \cup \{o\}) - f(A) \geq f(B \cup \{o\}) - f(B)$$

  This property encodes the diminishing returns of observations, i.e. making an additional observation is more valuable if the agents only have a few prior observations, than if they have made many. As discussed in Section 2.5.1, many observation value functions exhibit this property, such as entropy, mutual information, and area coverage.

  **Property 3** (Locality)**.** Observations taken sufficiently far apart in space are (almost) independent. That is, there exist a distance $\delta \geq 0$, and a $\rho \geq 0$, such that for any two sets of observations $A$ and $B$, if $\min_{a \in A, b \in B} \mathrm{d}(a, b) \geq \delta$, then:

  $$f(A \cup B) \geq f(A) + f(B) - \rho$$

  **Property 4** (Temporality)**.** Observations taken sufficiently far apart in time are (almost) independent. Formally, let $\sigma_t(\cdot)$ be a function that selects observations made at or after $t$:

  $$\sigma_t(A) = \{(v, t') \in A \mid t' \geq t\}$$

  Then, there exists a $\tau \geq 0$, such that for all $A \subseteq O$, $\epsilon \geq 0$:

  $$f(\sigma_{t-\tau}(A)) \geq f(A) + \epsilon$$

Additionally, for mobile agents the problem is augmented with the agents' motion constraints:

- A graph $G = (V, E)$ that represents the layout of the agents' environment, where edges $E$ encode the movements that are allowed between spatial coordinates $V$. Distance measure $\mathrm{d}_G(u, v)$ is the length of the shortest path in $G$.

For both fixed and mobile agents, each time step $t \in T$ proceeds in three steps:

1. Each agent $\mathcal{A}_i \in \mathbf{A}$ takes a set of observations $O_i^t \in O$. The number of observations depends on the domain and the type of events that occur within the environment. For example, when measuring temperature, an agent at some location $v$ is only capable of taking a temperature reading at $v$, and $O_i^t$ is a singleton. However, when scanning its environment for intruders, it might be able to observe locations other than $v$ alone, such that $O_i^t$ contains multiple observations.

2. Agents coordinate with their neighbours to determine which actions to take next. Each agent's neighbours are selected based on its communication range, but also, for example, on the locality parameter $\delta$, since there is no need for coordination if the value of observations are independent. This step is optional; at some time steps coordination might not be required.

3. Each agent takes an action. For mobile agents, this means moving to a location adjacent to its current one in graph $G$. For fixed agents, this might involve reorienting its viewing direction, or (de)activating itself.

The following examples illustrates the concepts introduced above for a scenario with mobile agents.[1]

**Example 3.1.** *Figure 3.1 shows four discrete time steps of the movement of four mobile agents. Here, it is assumed that the speed of the agents is sufficient to reach an adjacent vertex within a single time step. For illustration purposes, the observation value function $f$ assigns a value to observation at vertex $v$, which is equal to the number of time steps that have elapsed since $v$ was last observed, with a maximum of* 4*. Moreover, agents can only observe the vertex at which they are currently positioned.*

*The size of the vertices in Figure 3.1 are proportional to the observation value that can be received at their coordinates in the next time step. Thus, the value that the agents receive as a team in each of these four time steps are* 4*,* 12*,* 16 *and* 13*. Clearly, the decision by agent $\mathcal{A}_3$ to go back to its previous position (Figure 3.1(d)) results in a suboptimal observation value (at least, over these four time steps). If, instead, it had chosen to move to any other adjacent vertex, the observation value received by the team would have been 16.*

Now, as this example suggests, the goal of the agents is to maximise the observation value they receive as a team. To formalise this, we require some notation for describing different sets of observations:

[1] In this thesis, the term *mobile agent* is used to refers to an information gathering agent embedded in a mobile unmanned sensor. This is different from the common usage of this term, where is it used to refer to software agents that can migrate from one computer to another.

- The set of observations made by $\mathcal{A}_i$ at time $t$ is denoted as $O_i^t$.
- The set of observations made by all agents at time $t$ is denoted as $O_{\mathbf{A}}^t$.
- The set of observations made by all agents at *or before* time $t$ is denoted as $\mathbf{O}_{\mathbf{A}}^t$. By definition, for $t < 0$, $\mathbf{O}_{\mathbf{A}}^t = \varnothing$.

We will refer to the value received by making observations $O_{\mathbf{A}}^t$, given that observations $\mathbf{O}_{\mathbf{A}}^{t-1}$ where previously made, as their *incremental value* $\rho$:

$$\rho_{O_{\mathbf{A}}^t}(\mathbf{O}_{\mathbf{A}}^{t-1}) = f(O_{\mathbf{A}}^t \cup \mathbf{O}_{\mathbf{A}}^{t-1}) - f(\mathbf{O}_{\mathbf{A}}^{t-1}) \tag{3.1}$$

The agents' ultimate aim, to provide accurate situational awareness, can now be expressed in terms of maximising the discounted incremental value received during their mission:

$$\sum_{t \in T} \gamma^t \rho_{O_{\mathbf{A}}^t}(\mathbf{O}_{\mathbf{A}}^{t-1}) \tag{3.2}$$

Where $0 \leq \gamma \leq 1$ is a discount factor, that is used to control how much observations in the near future are worth compared to those in the longer future.

**Scope and Limitations** The properties of this problem have been carefully chosen to be as generic and realistic as possible, while at the same time offering several properties that can be algorithmically exploited. However, at this point, it is worth offering a brief discussion about the limitations of this formalisation. Specifically, these limitations can be broken down based on the three main properties of value function $f$:

**Submodularity** As discussed in Section 2.5.1, the submodularity of function $f$ implies that observations are subject to diminishing returns. Thus, this rules out the possibility of *synergy* between the agents. Synergy is found settings in which the value of observations is *superadditive*, such as those wherein certain sets of observations are valuable only if combined with others. Consider, for example, the case where the information collected about some feature of the world has to exceed a certain threshold to be of any value (see for example, Bian et al. (2006)). Suppose, furthermore, that this minimum amount of information is provided by a set of observations $S$. Thus, for any $A, B \subset S$, $A \cup B = S$, function $f$ is defined as $f(A) = f(B) = 0$, and $f(A \cup B) = v \geq 0$. Clearly, this function is not submodular. However, it is important to note that in the absence of such thresholds, two arbitrary sets of observations commonly contain a non-negative amount of *redundancy*. Therefore, in such cases, submodularity is a valid assumption.

| | **Chapter** | | | |
|---|---|---|---|---|
| **Property** | **4** | **5** | **6** | **7** |
| Submodularity | R | R | R | R |
| Locality | S | S | S | S |
| Temporality | O | O | S | S |

TABLE 3.1: Properties exploited or required by the algorithms presented in this thesis. 'R' means the property is required, 'S' that it is a soft requirement, and 'O' indicates that the property is not required.

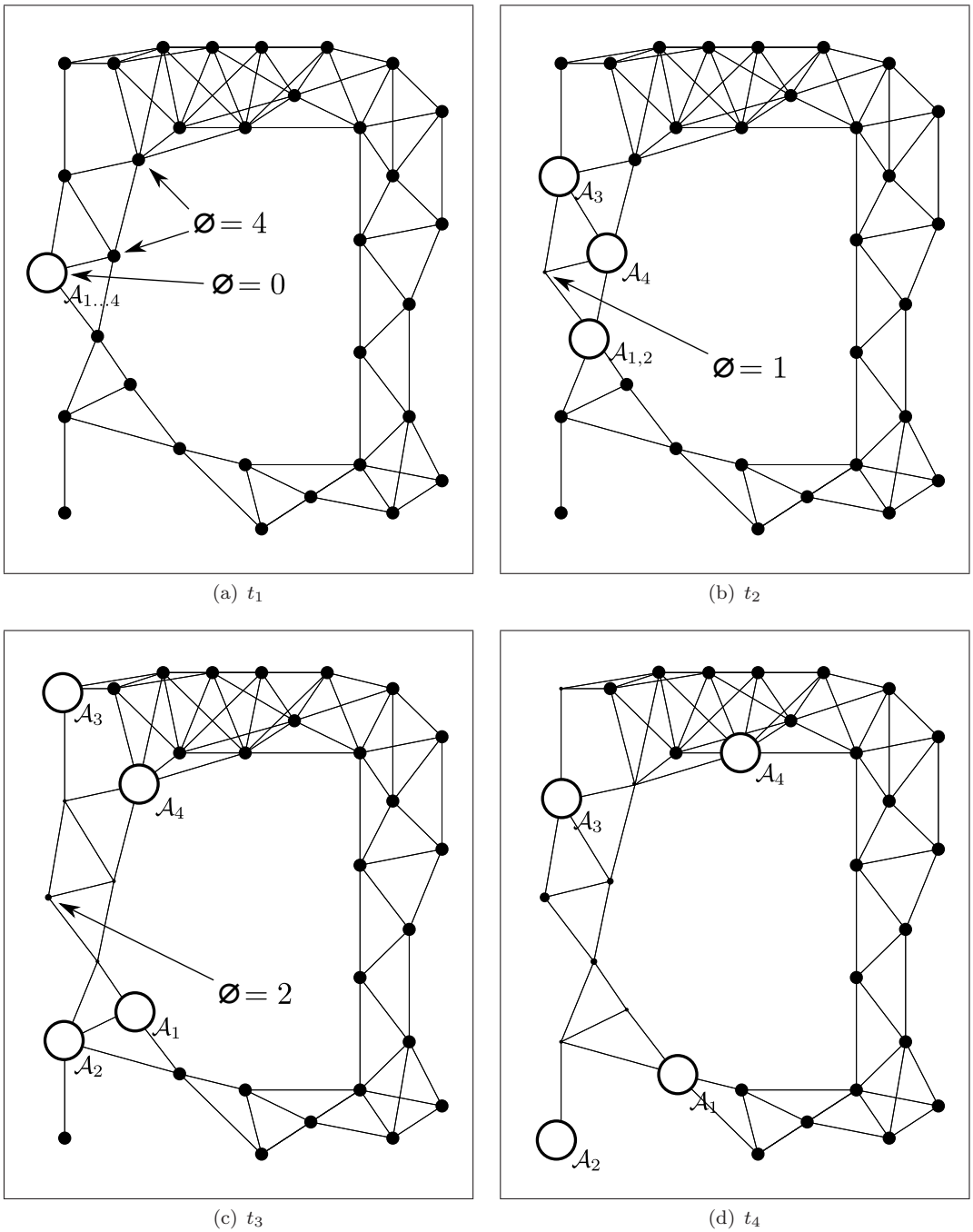
**Locality** Unlike submodularity, locality is not a binary requirement, but exhibits itself in degrees. At one extreme, $\delta = 0$, the value of observations is completely independent (i.e. additive), while at the other extreme, $\delta = \infty$, all pairs of observations are dependent (given a fixed $\rho$ for both extremes). In the former case, the problem can be decomposed into several subspaces of $L$, while in the latter case, the problem is such that a single observation provides information about all locations of $L$, and can therefore not be decomposed without losing important dependencies between the resulting subproblems. This leaves one final possibility: the existence of both local and non-local dependencies between sets of observations. Such problems do not exhibit locality, but can be approximated by taking the largest range $\delta$ outside which observations are guaranteed to be independent.

Now, since locality determines to a large extent the decomposability of the problem, it affects the computational overhead of the algorithms presented in the upcoming chapters. In particular, this overhead is a non-decreasing function of $\delta$. Locality can therefore be regarded as a *soft* requirement, since the scalability of the algorithms is adversely affected by lesser degrees of locality. As such, the requirement of locality does not necessarily limit the range of settings in which these algorithms can be applied, but can affect the quality of situational awareness (in the case that $\delta$ is chosen to be smaller than its true value) or their performance (if $\delta$ is large compared to the size environment).

**Temporality** Considerations similar to those for locality apply for the property of temporality. Strong temporality ($\tau = \infty$) implies that all observations taken in the past affect the value of observations taken now or in the future. Conversely, weak temporality ($\tau = 0$) indicates that the environment is changing so rapidly that observations taken during two consecutive time steps are completely independent. Clearly, realistic environments exhibit a degree of temporality that is somewhere in between these two extremes. Just as locality, temporality is a soft requirement: strong temporality means that observations need to be stored for a longer time, and the calculation of observation value needs to take into account a longer observation history. Again, since since this requirement is soft, assuming temporality does not limit the applicability of the algorithms presented in the upcoming chapters.

Not all of the properties discussed above are required or exploited. Table 3 gives a brief overview of the mapping between chapters and properties. More details are given in the chapters in question.

Even under these simplifying assumptions, solving this problem (and several varieties of it) optimally has been shown to be NP-hard (Meliou et al. 2007, Guestrin et al. 2005). Thus, in the next chapters, we propose several approximate algorithms that compute high *quality* and *scalable* solutions. These algorithms operate at different stages of the agents life cycle (i.e. before or after deployment), for both fixed and mobile agents. In the next chapter, we start with the first algorithm for observation value maximisation at the deployment stage of fixed agents.




(a) $t_1$ (b) $t_2$

(c) $t_3$ (d) $t_4$

FIGURE 3.1: Four discrete time steps of a team of mobile agents $\mathbf{A} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4\}$ moving in an environment whose layout is defined by a graph $G = (V, E)$. The diameter of the vertices (indicated by the $\varnothing$ symbol) is proportional to the observation value that is be received by moving there in the next time step.

# Chapter 4

# Decentralised Coordination for Fixed Agents during Deployment

In this chapter[1] and the next, we focus on the challenges of deploying and coordinating fixed information gathering agents. In addressing the challenge of deploying these agents, we develop an algorithm to construct a reliable communication network while at the same time maximising observation value. We then go on in the next chapter to discuss the use of decentralised coordination during their operation in order to further improve the agents' performance.

Now, as we have seen in Section 2.7, in order to achieve an acceptable level of quality with fixed agents (all other things being equal), their lack of mobility has to be compensated for by having a large number of them. In such large numbers, a system of agents resembles a network of micro-sensors or a wireless sensor network. Recently, these networks have generated a significant amount of interest in several of the areas we discussed in Chapter 2: climate change research (Padhy et al. 2006), weather and tidal surge prediction (Osborne et al. 2008, Kho et al. 2009), and monitoring intelligent buildings (Guestrin et al. 2005), to name a few. These networks consist of cheap sensors

<table>
<tr><th>Chapter</th><th>Type of Agent</th><th>Planning Horizon</th><th>Lifecycle Phase</th></tr>
<tr><td>4</td><td rowspan="2">Fixed</td><td rowspan="2">1</td><td>Deployment</td></tr>
<tr><td>5</td><td rowspan="3">Operation</td></tr>
<tr><td>6</td><td rowspan="2">Mobile</td><td>$m$</td></tr>
<tr><td>7</td><td>$\infty$</td></tr>
</table>

Table 4.1: The contributions of Chapter 4 in the context of the roadmap of this thesis.

[1] This chapter is based on Stranders, Rogers & Jennings (2010).

with very limited computational capabilities, which can potentially be deployed by scattering them from airplanes or ground vehicles. In the not-so-distant future, advances in miniaturisation could reduce the dimensions of these sensors to the micrometer scale, allowing them to be deployed as an aerosol. This idea was coined by Warneke et al. (2001), who refer to these sensors as *smartdust*.

Crucially, their limited size imposes restrictions on the computational resources they possess. Thus, agents embedded in these sensors need simple and robust algorithms for performing both the task of maximising observation value, as well as that of controlling various aspects of the wireless communication network. A particularly important aspect in this regard is the assignment of radio frequencies or, equivalently, time slots, for the Frequency or Time Division Multiple Access protocols (Bryan et al. 2007) that these sensors typically use, to minimise the number of retransmissions required due to interference. This is the problem we focus on in this chapter.

Several aspects of this problem have already been studied in the literature. In particular, when cast as a multi-agent graph colouring problem, the frequency assignment problem can be solved by the max-sum algorithm, as well as the various other message-passing algorithms that we discussed in Section 2.6.1. However, as we have seen, these algorithms either yield approximate solutions (e.g. max-sum, DSA), that do not rule out the possibility of interference; or require exponential computation and communication (e.g. DPOP), which pushes these algorithms beyond the limited computational abilities of the agents. Moreover, even if the agents were able to solve the frequency problem optimally, the inherent complexity of the frequency assignment problem often requires a large number of frequencies to prevent interference entirely. This means that total available bandwidth has to be divided into many segments, thereby significantly reducing the effective available bandwidth of the network (Bryan et al. 2007).

In this chapter, we therefore offer a novel approach to this problem. Instead of solving the graph colouring problem in the original network of agents, we develop a novel decentralised coordination algorithm that *deactivates* agents, such that the communication graph that exists between the remaining agents is more easily colourable. In so doing, the frequency assignment problem can be solved by simple and standard decentralised coordination algorithms. More specifically, our algorithm constructs a *triangle-free* graph; a graph that does not contain cliques greater than 2. This is appealing, because it is known that graphs with this property are 3-colourable (Thomassen 1994) in linear time (Dvořák et al. 2009). Equally important, we show that this limits the number of required frequencies to six, which would otherwise be very large for the original network of agents.

This, however, poses a new question. Which agents should be deactivated in order to ensure the communication graph is triangle-free, while at the same time maximising

observation value? We show that this modified problem is NP-hard (based on theoretical results from Nemhauser et al. (1978)), and that, therefore, no efficient optimal algorithm for this problem exists. Instead, we develop an approximate algorithm, which allows agents to coordinate in a fully decentralised fashion to construct a triangle-free communication graph and maximise observation value at the same time.

In more detail, the contributions of this chapter are:[2]

1. We derive a novel decentralised algorithm for solving the frequency assignment problem, which activates a subset of the available agents so as to maximise observation value, subject to the communication graph being triangle-free.
2. We develop a centralised greedy algorithm based on the concept of *submodular independence systems*, and derive a theoretical lower bound of $1/7$ on the approximation ratio of the algorithm, for any submodular function. This algorithm acts as a benchmark for the decentralised algorithm in the empirical evaluation.
3. We develop dynamic counterparts for these two algorithms that are capable of dealing with failing agents and newly introduced agents, while ensuring the triangle-free property of the graphs.
4. We empirically evaluate our algorithm and show that it provides 90% of the observation value provided by an optimal algorithm, and $> 60\%$ of the value provided by activating all agents (ignoring the frequency assignment problem). Moreover, we show that the frequency assignment problem on the agent network that results from applying our algorithms can be solved reliably by a simple and standard decentralised graph colouring algorithm. Finally, in the dynamic setting, we show that our algorithm provides 250% more observation value over time compared to activating all available agents simultaneously.

The remainder of this chapter is organised as follows. In Section 4.1 we extend the problem formulation from Chapter 3 to this problem. In Section 4.2 we present a centralised and a decentralised algorithm. In Section 4.3 we extend these algorithms to operate continually to replace failing agents. In 4.4 we empirically evaluate this set of algorithms and demonstrate their effectiveness. Finally, we give a summary of the contributions of this chapter in Section 4.5, and analyse them in terms of our design requirements.

[2]Table 4.1 shows the context of these contributions in terms of the roadmap of this thesis.

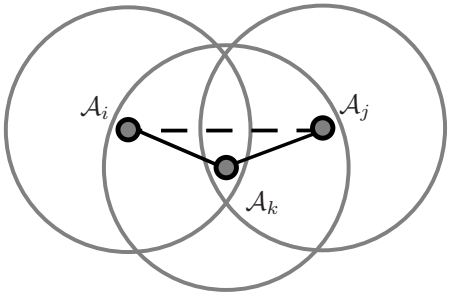



FIGURE 4.1: The communication and collision graph (solid) of an example agent network (solid and dashed). Possible direct and indirect collisions are represented by solid and dashed edges respectively.

## 4.1 Problem Description

In this section we define the problem that we address in this chapter. To do this, we extend the problem of maximising observation value, which was formulated in Chapter 3, with restrictions on the communication graph.

In more detail, let $\mathbf{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_M\}$ denote a set of $M$ agents deployed on the $\mathbb{R}^2$ plane. Their Cartesian coordinates are given by a vector $\mathbf{x}_i = (x_i, y_i)$. Let $d(\mathbf{x}_i, \mathbf{x}_j)$ denote the Euclidean distance between $\mathcal{A}_i$ and $\mathcal{A}_j$. Each agent $\mathcal{A}_i$ has a radio disk with radius $r_i$ within which other agents can receive its transmissions. Consequently agent $\mathcal{A}_j$ can receive $\mathcal{A}_i$'s transmissions iff $\mathcal{A}_j$ is contained within $\mathcal{A}_i$'s radio disk: $d(\mathbf{x}_i, \mathbf{x}_j) \leq r_i$. Each agent $\mathcal{A}_i$ has control over its transmission radius $r_i$, which it can set anywhere between 0 and $r_{max}$, which is the maximum transmission radius for all agents. Given this model, we can construct a *communication graph* that models the communication network that exists among the agents:

**Definition 4.1.** A *communication graph* $C[\mathbf{A}]$ of a set of agents $\mathbf{A}$ is a directed graph $C[\mathbf{A}] = (\mathbf{A}, E)$ in which $E$ contains a pair of agents $(\mathcal{A}_i, \mathcal{A}_j)$ if $\mathcal{A}_j$ can receive $\mathcal{A}_j$'s transmissions.

Now, to ensure transmissions between two agents are not compromised by interference from other agents, we wish to allocate frequencies to each agent such that no two agents with overlapping radio disks are allocated the same transmission frequency. However, note that the communication graph only models *direct* collisions—those that occur between $\mathcal{A}_i$ and $\mathcal{A}_j$ if they are contained within each other's radio disks. It does not model the possibility of *indirect* collisions (Bryan et al. 2007), which occur when two unconnected agents $\mathcal{A}_i$ and $\mathcal{A}_j$ transmitting on the same frequency, are in range of an agent $\mathcal{A}_k$ (see Figure 4.1 for an example). In this case, agent $\mathcal{A}_k$ will receive garbled transmissions from $\mathcal{A}_i$ and $\mathcal{A}_j$. To model these indirect collisions, we need to consider the *collision graph* $C^2[\mathbf{A}]$:

**Definition 4.2.** The *collision graph* of a set of agents $\mathbf{A}$ is the square of their communication graph $C[\mathbf{A}]$, denoted by $C^2[\mathbf{A}]$. This graph contains an edge $(\mathcal{A}_i, \mathcal{A}_j)$ if there exists a path between $\mathcal{A}_i$ and $\mathcal{A}_j$ in $C[\mathbf{A}]$ of at most two edges.

By effectively connecting neighbours of neighbours in the communication graph, the collision graph models the possibility of direct as well as indirect collisions. Thus, solving the frequency allocation problem is equivalent to colouring $C^2[\mathbf{A}]$, which is a well-known NP-complete problem (Karp 1972). Now, to bound the number of required colours needed to colour the graph (also referred as its chromatic number (Gross & Yellen 1999)), which can be very large in an arbitrary communication network, and to keep the algorithms as simple and robust as possible, we proceed in two steps.

First, we wish to find a set of agents whose communication graph $C[\mathbf{A}]$ is easily colourable, by ensuring it is *triangle-free*. A triangle-free graph is a graph that does not contain any cycles of length 3, or, equivalently, is free of cliques of size 3 and greater. A 3-colouring of a triangle-free graph is guaranteed to exist (Thomassen 1994), and can be computed in linear time (Dvořák et al. 2009). This colouring avoids any *direct* collisions. In the second step we attempt to avoid any *indirect* collisions by considering the denser *collision graph* of this triangle-free communication graph. Simple graph theory shows that this graph is guaranteed to be $K_7$ minor-free,[3] based on the triangle-free property of the communication graph. By exploiting this property, and applying the famous Hadwiger conjecture (Bollobás et al. 1980) we know that the obtained collision graph is 6-colourable.[4] Thus, the maximum number of colours needed to colour the collision graph of a triangle-free communication graph is 6. In Section 4.4, we show that this can be achieved by a simple decentralised graph colouring algorithm. Due to this correspondence between triangle-free communication graphs and their associated 6-colourable collision graph (Definition 4.2), in the remainder of this chapter, we will consider the communication graph only.

Second, besides ensuring reliable communication between the agents, we also wish to maximise the observation value that the agents receive. Defined as submodular in Chapter 3, this value is given by a submodular function of the agents' observations. Since the agents are fixed, the observation value they receive over their lifetime is dependent only on their location. In order to simplify notation, we slightly deviate from the problem formulation in Chapter 3, and define observation value function $f$ over the set of active agents, instead of the set of observations. As a result, the observation value received by

[3] A $K_7$ minor-free graph does not contain the complete graph $K_7$ as a subgraph, i.e. it contains no cliques larger than 6.

[4] The Hadwiger conjecture states that any $K_k$ minor-free graph is $(k-1)$-colourable, and has been proved for $k \leq 6$ (Robertson et al. 1993). In this chapter, we assume that the conjecture holds for $k = 7$, on the grounds that partial results are known for this case, and no counter example has yet been found. In more detail, it is known that graph with chromatic number 7 must contain either a $K_7$ minor graph, or both a $K_{4,4}$ and a $K_{3,5}$ (Kawarabayashi & Toft 2005).

a subset of $\mathbf{A}$ is given by a non-decreasing submodular function $f : 2^{\mathbf{A}} \to \mathbb{R}^+$, which accurately captures the diminishing returns of activating an extra agent.

Thus, the problem we address in this chapter is to find a subset of all available agents $\mathbf{A}' \subseteq \mathbf{A}$ that maximises $f(\mathbf{A}')$ subject to the communication graph $C[\mathbf{A}']$ being triangle-free. These agents $\mathbf{A}'$ will then form the deployed network of agents, and agents $\mathbf{A} \setminus \mathbf{A}'$ are deactivated.

These agents are later used to replace failing agents in a dynamic version of this problem. One major cause of agent failure is battery depletion. In this chapter, we assume that radio transmission accounts for the majority of the energy consumption of an agent, and thus we do not consider the energy required for sensing.[5] In more detail, agents have an initial energy supply $b_i$, which depletes over time as a result of their transmission power as follows: $\Delta b_i = -r_i^2 \cdot \Delta t$.

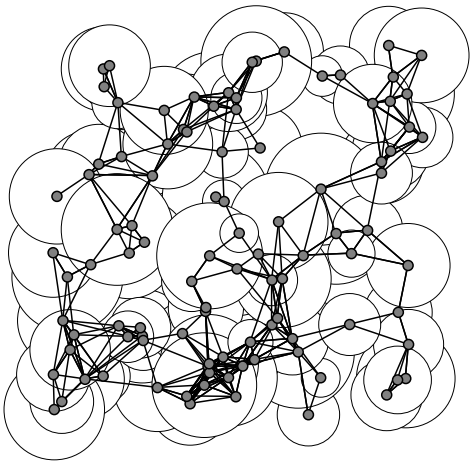
In the upcoming sections, we show how a simple and standard decentralised algorithm can be used to compute agent deployments $\mathbf{A}'$ that achieve high observation value, while ensuring the triangle-free property of the communication graph.

## 4.2 Two Deployment Algorithms for Fixed Agents

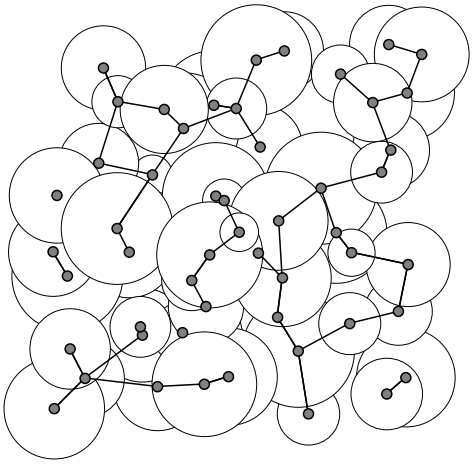
In this section, we develop two algorithms. The first is a centralised greedy algorithm with theoretical bounds on the solution. This algorithm will act as a benchmark for the second algorithm, which is a decentralised coordination algorithm. We then demonstrate that by deactivating agents, the algorithms are likely to break the communication graph into multiple unconnected components, thereby disrupting the flow of information between them, or to a base station. Therefore, in Section 4.2.3, we add a second phase to these algorithms, which is implemented by a decentralised algorithm that attempts to reconnect the various components of the graph by incrementally increasing the communication range of particular agents within the network.

To illustrate this two-phase approach, Figure 4.2 shows the output of the centralised algorithm for an example deployment with $M = 100$ agents. Figure 4.2(a) shows the original deployment. Here, the observation value received by the agents is the total area covered by their observation areas (the disks). Figure 4.2(b) shows the result of phase 1 of the algorithm. Notice how it selects a subset with high coverage, but also breaks the communication graph into 8 unconnected components. Figure 4.2(c) shows the effect of phase 2, which reconnects these components, while ensuring that the resulting communication graph remains triangle-free. The key feature of the resulting deployment in Figure 4.2(c) is that the communication graph is colourable with 3 colours, and the

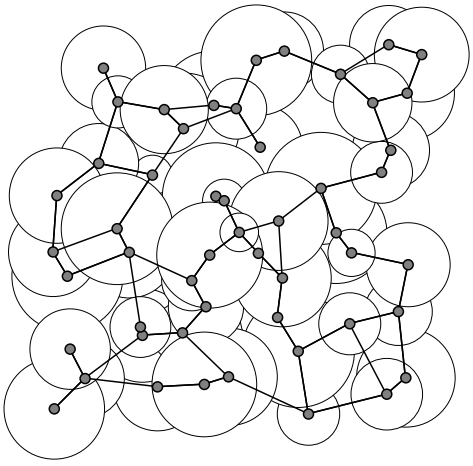
[5] For example, in GLACSWEB, a network of fixed sensors for monitoring glaciers, the energy expended on transmitting packets is 30 times greater than the energy expended on sensing (Padhy et al. 2006, Table 1).



(a) The original deployment of 100 agents.



(b) Agents selected by the centralised greedy algorithm.



(c) The communication graph after the reconnection phase.

FIGURE 4.2: Example execution of the two phase deployment algorithm. The circles represent the observation areas of the agents, the total area of which is proportional to the observation value they collectively receive. An edge between two agents indicates communication between them is possible.

collision graph with 6 colours, instead of the 23 colours required by the original graph in Figure 4.2(a), and the $\gg 23$ colours required to colour its collision graph. Moreover, due to redundant coverage in the original deployment, the agents selected by the algorithm provide almost the same amount of observation value as the agents in Figure 4.2(a).

### 4.2.1 A Centralised Greedy Algorithm

Before introducing the decentralised algorithm, we first develop a centralised greedy algorithm. This algorithm iteratively activates agents that most increase observation value without introducing a triangle. To derive a bound on the performance of this algorithm, we formulate this problem as maximising a submodular function, subject to an independence constraint. This independence constraint is formulated by the concept of an *independence system* from combinatorial optimisation:

**Definition 4.3.** (Calinescu et al. 2007) An *independence system* is a pair $(E, \mathscr{I})$, where $E$ is a finite set of elements, and $\mathscr{I}$ is a collection of subsets of $E$ such that if $A \in \mathscr{I}$ and $B \subseteq A$, then $B \in \mathscr{I}$. Sets in $\mathscr{I}$ are said to be *independent*.

Clearly, the set $\mathscr{I}_{\triangle\text{-free}}$ of subsets of $\mathbf{A}$ whose communication graph is triangle-free forms an independence system, since every induced subgraph of a triangle-free graph is triangle-free. Now, since not every subset is equal in terms of observation value, we augment these independence systems with the observation value function $f$:

**Definition 4.4.** A *submodular independence system* is an independence system together with a non-decreasing submodular set function $f$.

Unfortunately, the problem of finding the set $I^* \in \mathscr{I}$ such that $I^* = \arg\max_{I \in \mathscr{I}} f(I)$ is NP-hard (Nemhauser et al. 1978). Thus, under the assumption that P $\neq$ NP, there does not exist a polynomial time algorithm for computing $I^*$. As a result, to obtain solutions that scale well with the size of the number of agents, we have to resort to approximation. One of the simplest approximation algorithms is the greedy algorithm (Algorithm 2), that builds a solution without backtracking by iteratively adding those elements that most improve the solution (with respect to $f$), while simultaneously satisfying an independence constraint.[6] It is efficient, since it requires $O(|E|^2)$ computation steps (in each of the $|E|$, it computes $e^*$, requiring at most $|E|$ evaluations of $f$ in line 3).[7] Note that this algorithm is similar to the greedy algorithm discussed in Section 2.5.1, with the difference that the latter maximises a submodular function subject to a cardinality constraint.

[6] Note that in this algorithm, the independence system $\mathscr{I}$ need not be explicitly given. Typically, an oracle in the form of an algorithm or indicator function $\mathbf{1}_{\mathscr{I}}(\mathbf{A}) = \mathit{true} \Leftrightarrow \mathbf{A} \in \mathscr{I}$ suffices.

[7] See (Krause et al. 2008) for additional techniques to speed up the greedy algorithm.

**Algorithm 2** The greedy algorithm for a submodular independence system $((E, \mathscr{I}), f)$

```
1: I := ∅
2: while E ≠ ∅ do
3:    e* := arg max_{e∈E} f(I + e) − f(I)
4:    E := E − e*
5:    if I + e* ∈ 𝒥 then
6:       I := I + e*
7:    end if
8: end while
9: return I
```

The greedy algorithm computes a *maximal independent* set—an independent set $I$ that becomes dependent by adding any $e \in E \setminus I$. In other words, no agent can be added to the network without introducing a triangle.

Now, while the greedy algorithm is simple, it has its drawbacks; the main one being that it can perform arbitrarily badly, as illustrated by the following example:

**Example 4.1.** *Let $E = \{A, B_1, \ldots, B_M\}$, $\mathscr{I} = \{\{B_1, \ldots, B_M\}, \{A\}\}$. Moreover, let function $f$ be given by $f(\{A\}) = n$, $f(\{B_i\}) = n-\epsilon$, $f(\{B_1, \ldots, B_M\} = (n-\epsilon)M$. Thus, the result of the greedy algorithm is $I = \{A\}$ after a single iteration, whereas the optimal solution is $I^* = \{B_1, \ldots, B_M\}$. When $M \to \infty$ and $\epsilon \to 0$, the approximation ratio for Algorithm 2, i.e. $f(I)/f(I^*) = (n-\epsilon)/((n-\epsilon)M)$, approaches 0. In other words, for arbitrary independence systems, the greedy solution can be arbitrarily far from the optimal solution.*

Fortunately, many independence systems exhibit additional structure that can be exploited to obtain a lower bound on the approximation ratio for Algorithm 2, such as *p-independence* (Calinescu et al. 2007):

**Definition 4.5.** An independence system $(E, \mathscr{I})$ is called *p-independent* if for all $A \in \mathscr{I}$ and $e \in E$ there exists a set $B \subseteq A$ such that such that $|B| \leq p$ and $A \setminus B + e \in \mathscr{I}$.[8]

The following is a result in combinatorics that proves a lower bound on the approximation ratio of the greedy algorithm:

**Theorem 4.6** (Calinescu et al. (2007), Nemhauser et al. (1978))**.** *Algorithm 2 yields a $1/(1+p)$-approximation to maximising a non-decreasing submodular set function subject to a p-independence constraint.*

Thus, to obtain a lower bound on the observation value for the greedy algorithm, we need to determine $p$ for $(\mathbf{A}, \mathscr{I}_{\triangle\text{-free}})$. In order to do so, we need to restrict the problem defined in Section 4.1 slightly. This restriction involves limiting the radius of the radio

[8] When $p = 1$, a $p$-independence system is called a *matroid*, which is a well-known structure in combinatorics.

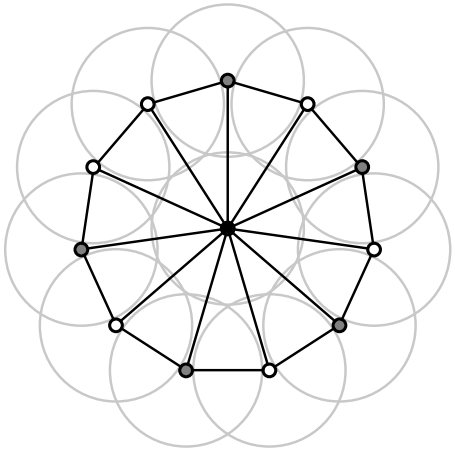


FIGURE 4.3: Visual representation of the proof of Theorem 4.7. See text for explanation.

disks to a constant $R$ for every agent.[9] A set of agents with a fixed radio range $R$ is denoted as $\mathbf{A}_R$, and the communication graph obtained is called a *unit disk graph*. This construct allows us to prove the following theorem:

**Theorem 4.7.** *System* $(\mathbf{A}_R, \mathscr{I}_{\triangle\text{-free}})$ *is 6-independent.*

*Proof.* Simple geometry shows that the maximum degree of a triangle-free unit disk graph is no larger than 11. Let $A$ be an valid solution (i.e. $A \in \mathscr{I}$). Now, $deg(e) \leq 11$ in $A + e$, otherwise $A \notin \mathscr{I}$. When $deg(e) = 11$, $A + e$ contains 11 triangles. To break each of these, we remove $p = \lceil 11/2 \rceil = 6$ vertices from $A$. Let $B$ denote this set of vertices. Then $A \setminus B + e \in \mathscr{I}$ with $|B| \leq 6$, as required. □

See Figure 4.3 for a visual representation of this proof. In this figure, $e$ is the black vertex, $B$ is represented by the white vertices and $A$ is represented by the white and grey vertices. Radio disks are represented by grey circles. (For ease of exposition, these radio disks have been scaled by 50%. As a result, links exists within this unit disk graph when the scaled disks overlap.)

As a result of Theorem 4.7, the greedy algorithm is guaranteed to produce a solution $I$ such that $f(I)/f(I^*) \geq 1/7$ for system $(\mathbf{A}_R, \mathscr{I}_{\triangle\text{-free}})$. However, we do not know whether or not this lower bound is tight. For example, note that in the worst case, greedy yields a 6/11 approximation on the construction used in the proofs.[10] Moreover, our empirical evaluation (see Section 4.4) obtained approximation ratios no less than 75%, even without the requirement that $r_i = R$ for all $i$, i.e. that all agents have identical radio ranges.

[9] We will drop this restriction again in our empirical evaluation, and show that this has no detrimental effect on the algorithm's performance.

[10] This occurs when $f(e) = n$, and for all $a \in A$, $f(a) = n - \epsilon$. The greedy selection of $e$ in the first iteration blocks the addition of 5 of 11 elements in $A$. Thus, the worst case approximation ratio is $\lim_{\epsilon \to 0} f(A \setminus B + e)/f(A) = {}^{6}/_{11}$.

### 4.2.2 A Decentralised Greedy Algorithm

The centralised algorithm assumes the existence of a centralised controller that has perfect knowledge of the locations of the agents, and the structure of their communication graph. Such centralised control is rarely possible in sensor networks, and thus, in the absence of a centralised controller, agents do not have access to the knowledge required to run the centralised greedy algorithm and determine if they should stay active. Moreover, while it is be possible to construct an algorithm that communicates this knowledge to all agents, its use would result in excessive communication between agents, which can not be considered scalable. In this section, therefore, we shall take a different approach and develop an approximate decentralised algorithm that relies on limited local communication and computation only. In more detail, this algorithm enables agents to construct a triangle-free network by ensuring their neighbourhood graph[11] is triangle-free. This is based on the simple fact that, if the neighbourhood of all sensing agents is triangle-free, the communication graph is triangle-free as well.

Against this background, the key principle behind the algorithm is that no more than two agents within the same clique can be activated without creating a triangle. The challenge is then to find which two agents should be activated to maximise observation value. Clearly, since solving this problem optimally in a central fashion is NP-hard, solving it optimally in a decentralised fashion is at least as hard. Therefore, we (again) resort to approximation in the form of a *greedy* decentralised algorithm. Using this algorithm, agents coordinate with their neighbours to determine if their activation blocks the activation of agents that provide more observation value. In more detail, when running this algorithm, an agent $\mathcal{A}_i$ continually checks whether a pair of agents ($\mathcal{A}_j$, $\mathcal{A}_k$) exist within the same clique, such that the observation value provided by this pair is greater than the value provided by either ($\mathcal{A}_i$, $\mathcal{A}_j$) or ($\mathcal{A}_i$, $\mathcal{A}_k$). If this is discovered to be the case, $\mathcal{A}_i$ is said to be DOMINATED. In all other cases, $\mathcal{A}_i$ is said to be DOMINATING. In the former case, the activation of agent $\mathcal{A}_i$ prevents the activation of $\mathcal{A}_j$ or $\mathcal{A}_k$, resulting in suboptimal sensing. Agent $\mathcal{A}_i$ should therefore deactivate itself. Conversely, in the latter case it is should stay active.

Algorithm 3 captures the necessary steps to determine the status of an agent. Before starting the main **while** loop, neighbours are discovered by means of message passing (lines 2—4).[12] Then, in lines 7 and 8, the agent attempts to find a non-DOMINATED neighbour that in turn has a non-DOMINATED neighbour in common with itself (i.e. a triangle). If no such neighbour can be found, the agent's best strategy is to activate itself (line 15), since no triangle is created. If, however, such a neighbour *does* exist, at least one of these three agents needs to deactivate in order to ensure that the graph is triangle-free. In line 11, the algorithm checks whether its activation blocks the activation

[11] The *neighbourhood* of a vertex is the subgraph induced by the vertex and its adjacent vertices.

[12] Naturally, since no collision free communication network exists at this stage, a simple collision detection and retransmission protocol can be used.

**Algorithm 3** The decentralised greedy algorithm for agent $\mathcal{A}_i$

1: $State \leftarrow$ Basic
2: $\mathbf{A}_i^B \leftarrow adj(\mathcal{A}_i)$ ($adj(\mathcal{A}_i)$ denotes the set of neighbours of $\mathcal{A}_i$)
3: Broadcast $\langle \mathbf{i}, \mathbf{A}_i^B \rangle$
4: Receive $\langle \mathbf{j}, \mathbf{A}_j^B) \rangle$ for all $\mathcal{A}_j \in adj(\mathcal{A}_i)$
5: **while** $State =$ Basic **do**
6: On random activation
7: $\mathbf{A}_i^B \leftarrow \{\mathcal{A} \mid \mathcal{A} \in \mathbf{A}_i^B \wedge State(S) \neq$ Dominated$\}$
8: **if** $\exists j : \mathbf{A}_i^B \cap \mathbf{A}_j^B \neq \varnothing$ **then**
9: Randomly select $\mathcal{A}_k \in \mathbf{A}_i^B \cap \mathbf{A}_j^B$
10: $f_{jk} \leftarrow f(\{\mathcal{A}_j, \mathcal{A}_k\})$
11: **if** $f_{jk} \geq f(\{\mathcal{A}_i, \mathcal{A}_j\})$ and $f_{jk} \geq f(\{\mathcal{A}_i, \mathcal{A}_k\})$ **then**
12: $State \leftarrow$ Dominated
13: **end if**
14: **else**
15: $State \leftarrow$ Dominating
16: **end if**
17: Send $\langle \mathbf{i}, State \rangle$ to all $\mathcal{A}_j \in adj(\mathcal{A}_i)$
18: **end while**

of two more 'valuable' agents. If this is found to be the case, the agent sets its state to Dominated, notifies its neighbours of its updated status, and deactivates itself (line 12).

Termination of this algorithm can be detected by inspecting the states of its neighbours: if all neighbours are either Dominated or Dominating, the algorithm has terminated. Termination of this algorithm is guaranteed; as the number of iterations approaches infinity, a Dominated agent will select a pair $(\mathcal{A}_j, \mathcal{A}_k)$ with probability 1 such that the condition in line 11 holds, and deactivate itself. All Dominating agents will remain in the Basic state, until all Dominated agents have deactivated themselves. At that point, Dominating agents will no longer be able to find a triangle (line 8), and thus detect their Dominating state (line 15).

In terms of complexity, note that it takes a Dominated agent $\mathcal{A}_i$ in expectation $\binom{m}{2}$ iterations of the **while** loop in Algorithm 3 (where $m = |adj(\mathcal{A}_i)|$ is the number of neighbours that $\mathcal{A}_i$ has) to find a pair $(\mathcal{A}_j, \mathcal{A}_k)$ that satisfies the condition in line 11, if such a pair is unique (and much less if there are multiple such pairs). A single execution of the loop takes $O(m \log(m))$ computation steps, which are required by computing the union of two sets in line 9. As a result, this algorithm is polynomial ($O(m^3 \log m)$) and thus efficient.

### 4.2.3 The Reconnection Phase

At this point we have developed two greedy algorithms that compute triangle-free subgraphs of $C[\mathbf{A}]$ with high observation value. However, in this process it is possible that

**Algorithm 4** The reconnection algorithm

1: **repeat**
2: Broadcast $\langle \mathbf{i}, adj(\mathcal{A}_i) \rangle$
3: Receive $\langle \mathbf{j}, adj(\mathcal{A}_j) \rangle$ for all $\mathcal{A}_j \in adj(\mathcal{A}_i)$.
4: $r_i \leftarrow c \cdot r_i$ {Increase communication radius}
5: **until** $adj(\mathcal{A}_i) \cap adj(\mathcal{A}_j) \neq \varnothing$ or $r_i > r_{max}$ {Until a triangle has been created}
6: $r_i \leftarrow r_i / \sqrt{c}$ {Break the triangle}

the communication graph is no longer strongly connected,[13] since these algorithms only ensure that the graph is triangle-free. To reconnect the components of the computed subgraph, we add a second phase to these algorithms. This algorithm is not always capable of fully reconnecting the graph, since the maximum radio transmission range $r_{max}$ is sometimes insufficient, but as we will show in Section 4.4, even when it is not completely successful, it manages to increase the size of the maximum connected component significantly.

Now, in an attempt to reconnect the graph, agents incrementally boost their radio signals to connect with agents that were previously unreachable. Clearly, it is undesirable for agents to set their radio strength to the maximum setting, for two reasons. Firstly, it will drain their battery quickly (at a rate of $r_{max}^2$), thereby reducing their lifetime. Secondly, by doing so, an agent is likely create edges in the communication graph that compose a triangle, which was exactly what we set out to avoid. To prevent this, each agent executes Algorithm 4 in order to connect to additional agents whilst maintaining a triangle-free graph in a fully decentralised fashion.

The algorithm operates by gradually increasing an agent's communication range (line 4), until it is discovered that a triangle has been introduced (lines 2, 3 and 5). At that point, the agent reduces its communication range to break it again (line 6). Thus, Algorithm 4 maximises $r_i$, while maintaining the triangle-freeness of the graph.

## 4.3 Dealing with Dynamism

In the previous section, we presented two algorithms that perform a one-off optimisation procedure for simplifying frequency assignment and maximising observation value. In this section, we consider a more dynamic setting, in which deployed agents can fail and new agents can be deployed. Now, as the example in Figure 4.2 and the experimental results in Section 4.4 show, the number of agents needed by both the centralised and decentralised algorithms is fairly small compared to the number of deployed agents. The remaining agents lie dormant and do not perform any useful tasks. However, in

[13] A graph is *strongly connected* if there exists a path from every vertex to every vertex. For an agent communication graph, this means that every agent is capable of communicating with all other agents via multi-hop routing. In the remainder of the chapter, when we use the term *connected* we mean *strongly connected*.

the dynamic setting, they can be used to replace failed agents. In this section, we show how this can be achieved, and we develop two dynamic counterparts of Algorithm 2 and Algorithm 3. Both algorithms continuously monitor the network and select replacements for agents that stop functioning. These dynamic algorithms are obtained as follows:

**Centralised:** Once an agent fails, Algorithm 2 is re-run, with $I$ initialised to the activated agents, minus the failing agents. The algorithm will then proceed to iteratively add new agents (if possible). When new agents are deployed, they are added to $E$, and the algorithm is run without modifications.

**Decentralised:** Instead of completely turning off DOMINATED agents, these agents lie dormant while monitoring communication in their neighbourhood. Once a neighbouring agent fails (which can be detected by a prolonged interval of communication silence), it resets its state to BASIC, and re-runs Algorithm 3. Active agents (i.e. those with a DOMINATING state) need not re-run the algorithm. Newly deployed agents set their state to BASIC, and run the algorithm without modifications.

In the next section, these dynamic algorithms are benchmarked against each other to determine their achieved solution quality over time.

## 4.4 Empirical Evaluation

We now evaluate the algorithms developed in the previous sections in a large scale agent deployment scenario. In the first part of the empirical evaluation, we measure the performance of the centralised and decentralised greedy algorithms (Section 4.4.2). In the second part, we subject the dynamic versions of these algorithms to empirical evaluation (Section 4.4.3). First, however, we describe the experimental setup common to both.

### 4.4.1 Experimental Setup

We consider a wide area surveillance scenario (Section 2.1.2) in which $M$ agents have been randomly deployed in a unit square, and are tasked with detecting events. The observation area $\bigcirc_i$ in which agent $\mathcal{A}_i$ can detect events is a disk with radius $s_i$, which is drawn from the interval $[0.05, 0.2]$ with uniform likelihood. The radius $r_i$ within which agent $\mathcal{A}_i$ can receive and send transmissions is uniformly drawn from the interval $[0.5R, R]$, where $R$ controls the range of $r_i$, and is one of the parameters we vary in our experiments. Moreover, the maximum radio transmission range is $r_{max} = 1.2R$. Events occur randomly within the unit square with uniform probability. The observation value

$f(\mathbf{A}')$ received by a subset $\mathbf{A}' \subseteq \mathbf{A}$ is the expected number of detected events, i.e. those that fall within the sensing disk of at least one agent $\mathcal{A} \in \mathbf{A}'$.

More formally, let $\bigcirc_i \subset \mathbb{R}^2$ denote the observation area of agent $\mathcal{A}_i$. Moreover, let $\mu(\cdot)$ denote the measure[14] of an area. Now, function $f$ is defined as:

$$f(\mathbf{A}) = \mu\left(\bigcup_{\mathcal{A}_i \in \mathbf{A}} \bigcirc_i\right) \tag{4.1}$$

Figure 4.2 shows an example of function $f$. The observation value obtained by the activated agents is equal to the total area covered by their observation areas.[15] Clearly, $f$ is a non-decreasing submodular function, since adding an agent $\mathcal{A}_i$ to a deployment $\mathbf{A}$ increases observation value less than adding the same agent to a smaller deployment $\mathbf{A}' \subset \mathbf{A}$.

### 4.4.2 Evaluation of the Greedy Algorithms

In the first set of experiments, we applied the centralised and decentralised greedy algorithms of Section 4.2 on 500 randomly generated deployments with $M = 300$. We varied $R$ between 0.1 and 0.5 to generate graphs with increasing density, which effectively controls the constrainedness of the problem: with $R = 0.5$, the communication range of the agents spans half the size of the environment, the communication graph contains many triangles.

We benchmarked the algorithms against an optimal algorithm for this problem, i.e. one that computes a triangle-free subgraph with optimal observation value ($I^*$ defined in Section 4.2.1). This algorithm uses branch and bound and exploits the structure of submodular functions to improve computational efficiency. Despite these computational efficiency improvements, however, such an optimal approach does not scale beyond $\approx 30$ agents.[16] Because of this, we performed two batches of experiments. In the first, we used 30 agents and evaluated the centralised, decentralised and optimal algorithms, and in the second batch we applied the centralised and decentralised algorithms on a deployment of 300 agents.

We used the following metrics to determine the performance of the algorithms:

[14] To avoid confusion, we use the term 'area' to describe two dimensional shapes, and 'measure' to denote the extent (or size) of an area.

[15] Note that this function assigns equal value to observations across the entire space. As such, the impact of observations on the quality of situational awareness is assumed to be homogeneous across the spatial dimension (see Definition 2.3). However, it is easy to weigh certain areas based on their importance, or the expected loss for not observing these areas. By so doing, submodularity is not violated.

[16] In more detail, on many problem instances, the optimal algorithm took $>$ 2 hours, while both greedy algorithms always terminated in less than 5 seconds on a standard desktop computer.

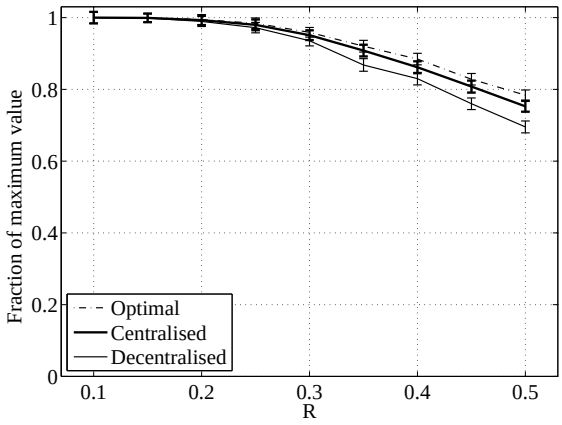
- The observation value received by the active agents normalised against the observation value received by activating all agents (i.e. by ignoring the constraints on the network).
- The observation value received by the largest connected component of the graph. This metric captures the trade-off between the graph connectedness and observation value, and is used to determine the effectiveness of the reconnection algorithm. More specifically, it is the observation value received in the most favourable case wherein a base station (a special node that collects data and transmits data for further analysis) is connected to one of the agents in this component.
- The number of active agents. This measures how well the algorithms are able to satisfy the constraints imposed by the triangle-free property, and shows how many agents are available to replace failed ones by the dynamic algorithms.

The results of the first batch ($M = 30$) are summarised in Figure 4.4. Figure 4.4(a) shows the observation value as a fraction of the observation value received by all $M$ agents. This plot clearly shows that the difference between the optimal solution and the solution computed by both greedy algorithms is less than 10% in the most constrained case (i.e. $R = 0.5$). This is a clear indication that both greedy algorithms compute very good approximations, without the need for exhaustively searching the solution space.

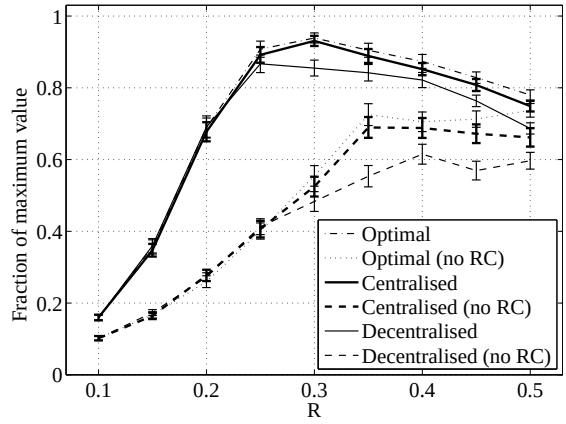
Figure 4.4(b) shows the observation value achieved by the largest component. In this figure, the postfix 'no RC' indicates that the reconnection algorithm from Section 4.2.3 was not used. This figure demonstrates the effectiveness of the reconnection algorithm. It manages to connect a sufficient number of components to almost double the observation value received by the largest component of the graph.

Finally, Figure 4.4(c) shows that the optimal algorithm manages to select a slightly larger number of agents compared to both greedy algorithms. As expected, both greedy algorithms are less successful in satisfying the independence constraints while maximising observation value. However, this effect is only marginal, since the optimal algorithm activates just 10% more agents than the decentralised greedy algorithm.

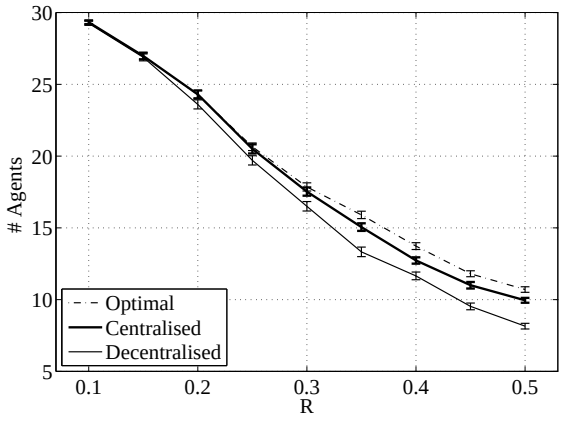
The results of the second batch (with $M = 300$) are shown in Figure 4.5. Overall, the same behaviour as before can be observed here. However, Figure 4.5(a) shows that the achieved observation value of the decentralised algorithm drops below 60% of the maximum achievable observation value for $R = 0.5$. The same, albeit less strong, effect can be observed for the centralised greedy algorithm. However, it is important to note that, with $R = 0.5$, the agents are able to communicate with agents in a quarter of the entire area. As a result, the communication graph of the original agent network is very dense, such that the problem of finding a triangle-free graph is very constrained. Figure 4.5(b) again demonstrates the effectiveness of the reconnection algorithm, but also that between $R = 0.1$ and $R = 0.3$ both algorithms provide at least 75% of the maximum




(a) Achieved fraction of maximum possible observation value.

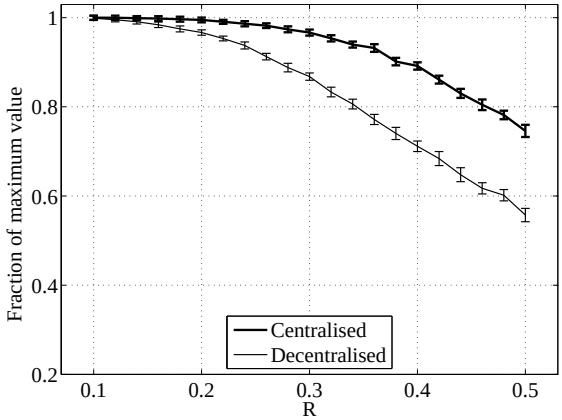



(b) Achieved fraction of maximum possible observation value by largest connected component.

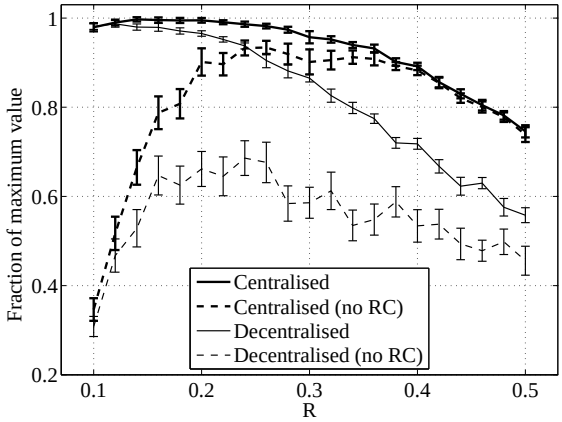



(c) Number of active agents.

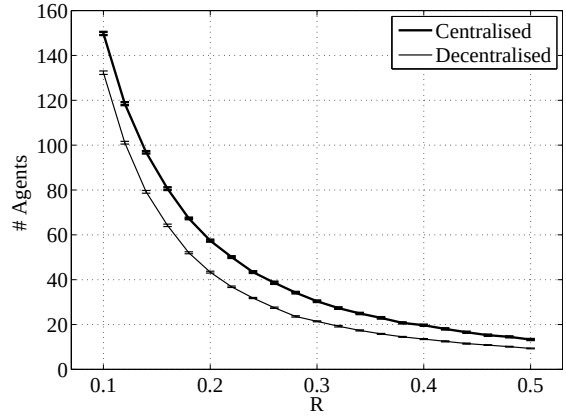
FIGURE 4.4: Empricial results for the static algorithms ($M = 30$). Error bars indicate the error of the mean.




(a) Achieved fraction of maximum possible observation value.




(b) Achieved fraction of maximum possible observation value by largest connected component.




(c) Number of active agents.

FIGURE 4.5: Empirical results for the static algorithms ($M = 300$). Error bars indicate the error of the mean.

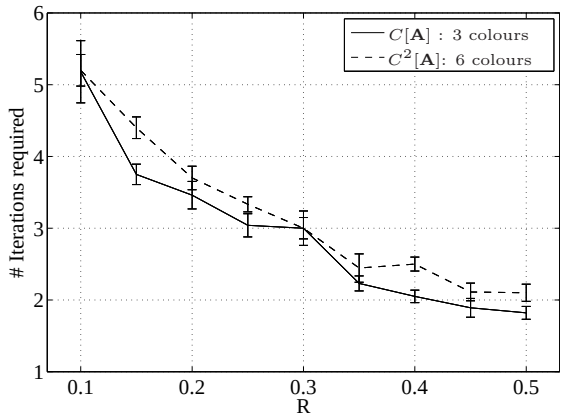



FIGURE 4.6: Iterations required by the greedy graph colouring algorithm ($M = 300$).

possible observation value, while needing approximately half (for $R = 0.1$) to a tenth for ($R = 0.3$) of the available agents.

Finally, to corroborate the theoretical result that the resulting communication and collision graphs are easily 3 and 6-colourable respectively (see Section 4.1), we used a simple and standard algorithm to colour 5000 graphs in a decentralised fashion. This algorithm is an $\epsilon_n$-greedy algorithm, i.e. with probability $1 - \epsilon_n$ an agent selects the colour that minimises the number of adjacent mono-chromatic edges, while with probability $\epsilon_n$ it picks a random colour. Furthermore, probability $\epsilon_n$ decreases with each iteration of the algorithm as $\epsilon_n \leftarrow \frac{n-1}{n}\epsilon_n$, where $n$ is the number of completed iterations. Figure 4.6 shows that colouring the resulting graphs in a decentralised fashion is indeed trivial. With 300 agents, the algorithm needs 5 iterations on average to correctly colour both the communication graph, as well as the collision graph. Moreover, this simple algorithm managed to find a colouring in all problem instances that we considered.

### 4.4.3 Evaluation of the Dynamic Algorithms

In the second experiment, we evaluated the dynamic algorithms. We simulated a randomly deployed network of agents as before, but we also considered the possibility of agents failing. In our simulations, agents are deactivated when their battery is depleted. Initially, every agent $\mathcal{A}_i$ has a battery capacity $b_i$ of 1 unit. Recall from Section 4.1 that the battery depletion rate is modelled as $\Delta b_i = -r_i^2 \cdot \Delta t$. Each time an agent fails, we run the algorithms developed in Section 4.3 to determine whether it can be replaced by agents that were not selected for the initial deployment. We benchmarked our algorithms against a naïve strategy (referred to as 'On') that activates all agents without considering any restrictions on the communication graph.

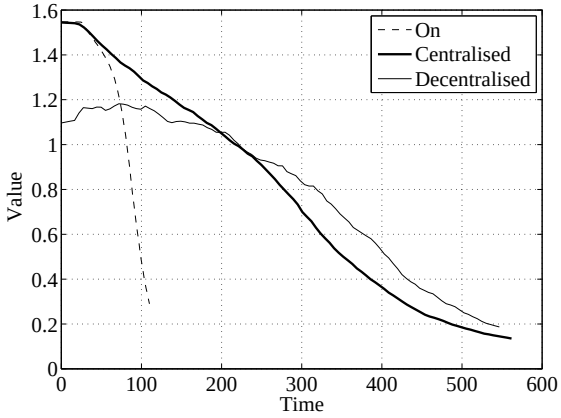
The results are shown in Figure 4.7. The plots in Figure 4.7(a) and 4.7(b) show the observation value over time achieved by all active agents and the largest component respectively. As can be observed in this figure, the observation value received by the

'On' strategy decreases rapidly. This is caused by the fact that all agents are active simultaneously, fail early, and waste their sensing area by redundantly covering the area. Compared to 'On', our algorithms perform notably better by providing coverage until well after $t = 500$. It is also worth noting that, whereas the decentralised algorithm is outperformed by the centralised one for the initial deployment (cf. Figures 4.5(a) and 4.5(b)), the decentralised algorithm starts outperforming its centralised counterpart after $t \approx 250$. The explanation for this is found in Figure 4.7(c) that shows the number of active agents over time. The decentralised algorithm requires less agents for the initial deployment, and therefore has more agents available to replace failed ones.

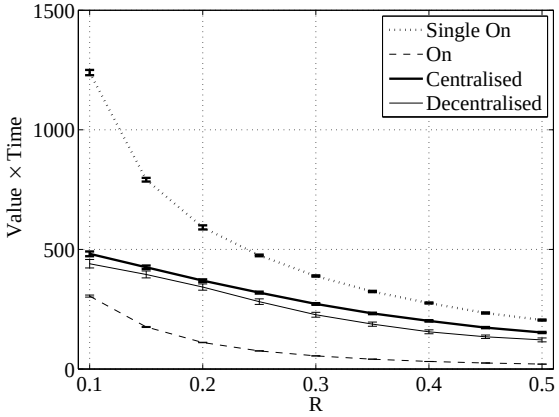
Finally, we recorded the total observation value received over time for several radio ranges $R$. Observation value over time is defined as the area of the region below the graphs shown in Figures 4.7(a) and 4.7(b). For this experiment, we added an additional benchmark strategy that activates only a single agent at a time (referred to as 'Single On'). The performance of this strategy acts as an upper bound on the maximum achievable performance, since no two agents redundantly cover the space. Figures 4.8(a) and 4.8(b) show the results. These figures confirm that 'On' is outperformed by both greedy algorithms for several values of $R$, and by around 250% for $R = 0.2$. Moreover, by comparing the performance of our algorithms to that of 'Single On', we can conclude that these algorithms quite effectively manage to minimise redundant coverage, since 'Single On' has no redundant coverage by its very nature. Most importantly, we can conclude that the decentralised algorithm achieves at least 80% of the sensing quality of the centralised greedy algorithm (92% for $R = .2$), while only requiring local communication and computation.

## 4.5 Summary

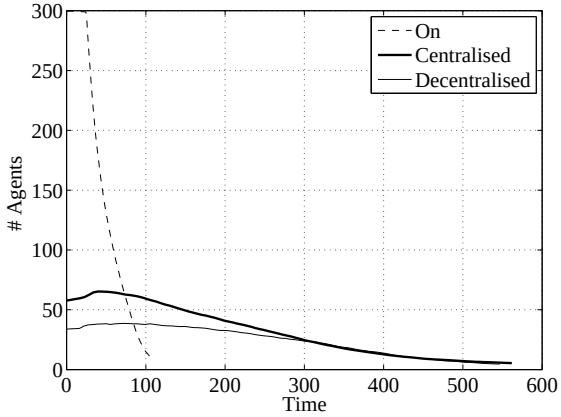
In this chapter, we developed a novel decentralised algorithm for simplifying the frequency assignment problem in a network of fixed information gathering agents, while at the same time maximising observation value. In particular, this algorithm activates a subset of agents, such that the communication graph is triangle-free. This way, only six frequencies are needed, and the frequency assignment problem can be solved using a simple decentralised graph-colouring algorithm. We also developed a centralised greedy algorithm based on the notion of submodular independence systems, for which we derived a theoretical lower bound of 1/7 on the approximation ratio. We then proceeded to consider dynamic settings, in which agents can fail, or new agents can be added to the existing deployment, and extended both our algorithm and the centralised greedy algorithm to operate in such settings. We empirically evaluated our algorithm by benchmarking it against the centralised algorithm and an optimal one. We showed that, in the most constrained case ($R = 0.5$), the selected agents manage to achieve 90% of the observation value received by the optimal algorithm, and 60% of the observation value




(a) Observation value received over time

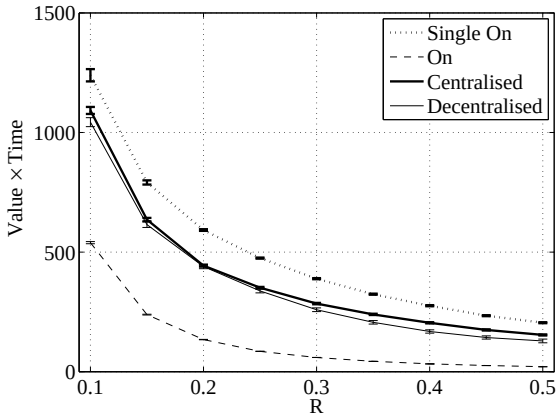



(b) Observation value received over time by the largest component

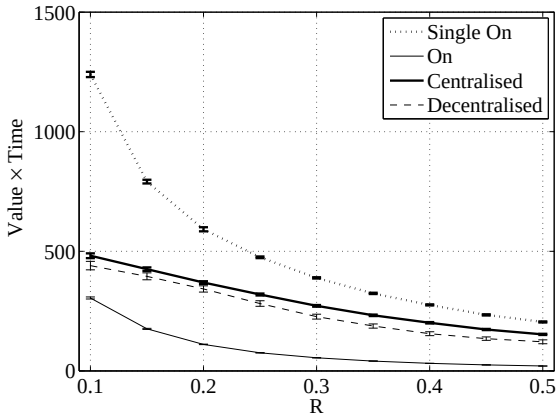



(c) Active agents over time

FIGURE 4.7: Results for the dynamic algorithms ($M = 300$, $R = 0.2$).




(a) Entire system.




(b) Largest component only.

FIGURE 4.8: Total observation value over time received by the dynamic algorithms.

received in the unconstrained case (by activating all agents). Finally, in the dynamic setting, our algorithm provides 250% more observation value over time compared to activating all available agents simultaneously.

In terms of the design requirements stated at the start of this thesis, we can conclude the following:

**Quality:** The empirical results from Section 4.4 show that the decentralised algorithm performs close to optimal ($> 90\%$) and provides more than 250% observation value compared to activating all agents simultaneously.

**Adaptiveness:** Both the centralised and decentralised algorithm compute a deployment under the assumption that observation value function $f$ does not change over the lifetime of the agents. If an *a priori* unknown event radically changes $f$ the deployment's performance can degrade significantly. The algorithms could be modified to detect such an event and re-run the algorithms from scratch, but in their current form they are not adaptive.

**Robustness:** A system of fixed agents that uses the dynamic decentralised algorithm is robust against failure, or will at least degrade gracefully, since it is capable of replacing failed agents with dormant ones. Moreover, using any of the algorithms developed in this chapter, the frequency assignment problem can be solved optimally, thereby ensuring the reliability of the communication network.

**Autonomy:** The decentralised greedy algorithm has no centralised components, and thus ensures the autonomy of the agents.

**Scalability:** The complexity of the centralised algorithm is quadratic in the number of agents ($O(|\mathbf{A}|^2)$), and the complexity of the decentralised algorithm for a single agent is $O(m^3 \log m)$, where $m$ is the number of neighbours. Thus, both algorithms are efficient (i.e. polynomial) and scale well in the number of agents.

**Modularity:** Although not specifically addressed in this chapter, the different capabilities of the agents can be encoded into the observation value function $f$. For example, agents that provide noisy coverage over a small area (for example, an infrared sensor) will provide less observation value than agents capable of monitoring a large area with high resolution (for example, a radar). We have already given an example of this when we assigned agents heterogeneous observation areas in Section 4.4.

**Performance Guarantees:** The centralised algorithm provides a 1/7 performance guarantee. Whether the decentralised algorithm provides similar performance guarantees is an open question, and is subject of future investigation.

In the wide area surveillance scenario we considered in this chapter, we assumed that agents can detect events within their observation range at all times. Under this assumption, no further coordination is needed after the agents are deployed using the algorithms developed in this chapter. However, if their operation is somehow constrained, for example because agents have a limited power supply, or can only observe one chosen segment of their observation disk at a time, it often pays off to coordinate their actions after their deployment as well. Therefore, in the next chapter, we develop a decentralised coordination algorithm that facilitates coordination at this stage of the agents life cycle. This algorithm should by no means be considered as mutually exclusive with the one we developed in this chapter, but rather as complementary. The latter can be used after the former to further improve solution quality.

# Chapter 5

# Decentralised Coordination for Fixed Agents during Operation

Once a group of fixed agents has been deployed—using the algorithms developed in the previous chapter or otherwise—they face the challenge of coordinating their actions at "runtime" in order to maximise the observation value they receive as a team. In the previous chapter, we assumed that this value depends on the agents' location only. This is a reasonable assumption at deployment time, when the agents only have knowledge of some statistical properties of the events that will occur in their environment, such as their spatial distribution or a probability distribution over their types. During their operation, however, when they can react and adapt to actual observed events, their actions, rather than their locations alone, primarily affect the observation value they receive. Careful coordination of these actions is thus essential to maximise the agents' performance.

To study these coordination problems, we consider two realistic wide area surveillance scenarios in which agents have fine grained control over the observations they make after deployment, thus making coordination between them essential:

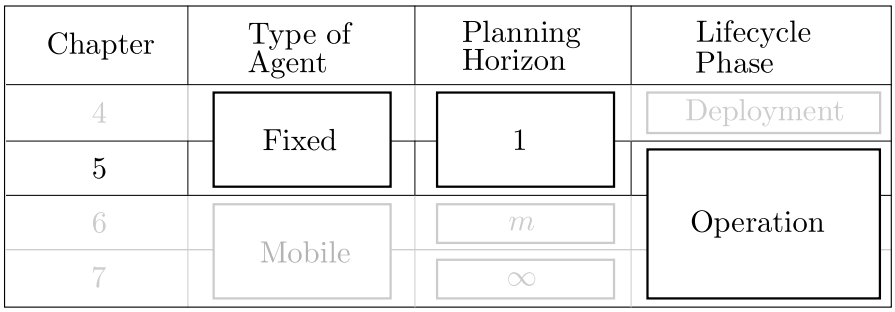
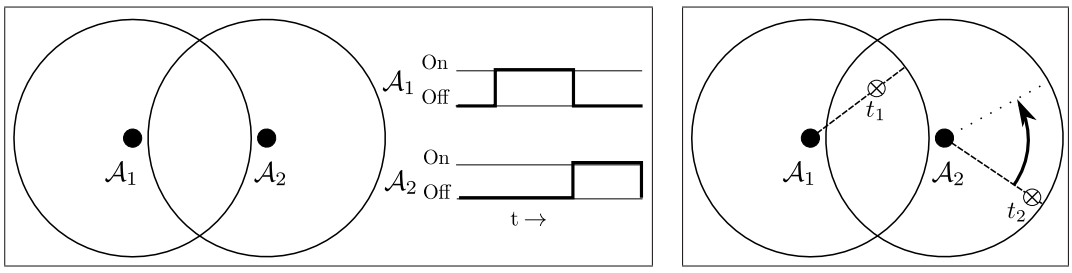



Table 5.1: The contributions of Chapter 5 in the context of the roadmap of this thesis.




(a) The event detection scenario.

(b) The target classification scenario.

FIGURE 5.1: Two examples of the wide area surveillance domain.

**Event Detection** This scenario is very similar to the one we considered in the previous chapter. However, instead of being able to monitor their observation area continuously, the agents are energy constrained; they are capable of harvesting energy from their environment, but at a rate that is insufficient to allow them to be powered continually. Thus, they need to regularly power down to conserve energy. However, once they power down, agents no longer observe their environment. Thus, agents whose observation areas overlap should coordinate over their activation schedules to maximise the probability of detecting events occurring within these overlapping areas. Figure 5.1(a) shows an example of this domain. Here, agents $\mathcal{A}_1$ and $\mathcal{A}_2$ have coordinated to ensure the intersection of their observation areas is monitored as long as possible, and the probability of detecting events occurring in it is maximised.

**Target Classification** In the second scenario, agents are tasked with classifying targets in their observation areas. Every agent is equipped with a sensor whose viewing orientation they control. By directly aiming these sensors at a target, they maximise the probability of correctly classifying targets. Since their observation areas are likely to overlap, agents should coordinate with their neighbours to maximise the number of targets observed by at least one agent. Figure 5.1(b) shows an example of this domain. Here, agents $\mathcal{A}_1$ and $\mathcal{A}_2$ have coordinated to maximise the probability of correctly classifying targets $t_1$ and $t_2$, which in this case is achieved by $\mathcal{A}_1$ observing $t_1$ and $\mathcal{A}_2$ observing $t_2$.

These scenarios share an important characteristic; the agents' decision variables have *continuous* domains. The activation schedules in the first scenario are set in continuous time, and the viewing orientation in the second scenario is expressed in radians, which vary between 0 and $2\pi$. Clearly, continuous action variables not only occur in these domains. Indeed, many other multi-agent coordination problems take place over continuous variables. For example, the problems of controlling multiple UAVs (whose control variables include pitch, roll, yaw and thrust), and controlling flows in a network (internet traffic, electricity grids, etc.), are both governed by variables with continuous domains.

The max-sum algorithm would normally be our algorithm of choice to solve decentralised coordination problems, since its properties are well aligned with our design requirements (as discussed in Section 2.6.1). However, in its original form, it is limited to settings where the agents' action variables are discrete. Thus, should we wish to apply it to solve the aforementioned continuous coordination problems, we must first discretise the continuous domains of the variables. This, however, is not without problems. On the one hand, care must be taken to ensure this discretisation is sufficiently coarse to keep the computation tractable, since the computation an agent needs to perform is exponential in the cardinality of its variable domain, as well as that of their neighbours. On the other hand, the discretisation must be sufficiently fine to enable the algorithm to find high quality solutions. This is especially true if the global function has sharp peaks[1], in which case the penalty for too coarse a discretisation can be high. Thus, the use of the standard max-sum algorithm (and, indeed, any other DCOP algorithm discussed in Section 2.6.1) in continuous domains involves a trade-off between the *quality* and *scalability* of the solution—two of the key design requirements of this thesis. A trade-off, crucially, that is not inherent to the problem.

In light of this, we identify a need for decentralised coordination algorithms that have scalable computation and communication costs, and compute good quality solutions for continuous problems. It is this specific need that we address in this chapter, and to this end, we present two extensions[2] to the max-sum algorithm which are the first two algorithms for distributed constraint optimisation problems (DCOPs) with continuous variables. These two extensions differ in the type of utility function they support. The first extension, max-sum for continuous *piecewise linear* functions, or CPLF-MS, provides an exact implementation of the two key mathematical operations used by max-sum for settings with piecewise linear utility functions. The second extension, hybrid continuous max-sum, or HCMS, combines the standard (discrete) max-sum algorithm with non-linear optimisation techniques, and is applicable to settings with *non-linear* continuous utility functions.

These algorithms are domain independent, and are thus not only applicable to the two coordination problems we consider in this chapter. However, to evaluate their performance, we apply these algorithms to solve the event detection and target classification problems, and show that this yields solutions that are highly desirable in terms of the design requirements of this thesis. In more detail, since the interactions between agents in the event detection scenario are characterised by linear utility functions, and thus apply the CPLF-MS algorithm. Similarly, we apply the HCMS algorithm to the target classification scenario where the utility functions are highly non-linear.

As a result, in this chapter we make two sets of contributions:[3]

[1] More formally, if the magnitude of the first and second derivatives of the global function are large.
[2] These two extension were published as Stranders et al. (2009*a*) and Voice et al. (2010) respectively.
[3] Table 5.1 shows the context of these contributions in terms of the roadmap of this thesis.

**CPLF-MS:** Max-sum for continuous piecewise linear utility functions:

1. We derive a representation of piecewise linear utility functions using simplexes. Then, using techniques from computational geometry, we develop exact algorithmic solutions for the mathematical operations required to apply the max-sum algorithm in the domain of continuous variables and piecewise linear utility functions; specifically, addition and marginal maximisation of general n-ary piecewise linear functions.
2. We empirically evaluate the performance of CPLF-MS in the event detection scenario where the activation schedules of agents need to be coordinated in order to maximise the system-wide probability of detecting events. We compare CPLF-MS against standard discrete max-sum, and show that it outperforms discrete max-sum (with up to a 10% increase in solution quality), with lower coordination overhead in terms of message size (the CPLF-MS yields a reduction of up to half the message size).

**HCMS:** Max-sum for non-linear continuous utility functions:[4]

1. We propose the hybrid continuous max-sum algorithm (HCMS), which combines the standard max-sum algorithm with continuous non-linear optimisation methods. For problems with acyclic factor graphs, we derive theoretical optimality results for this algorithm. In particular, we can show that, for suitable parameter choices, the HCMS algorithm outperforms the discrete max-sum algorithm operating over the same discretisation of the state space and, for sufficiently fine discretisations, the HCMS algorithm converges to a near optimal solution.
2. We empirically evaluate our HCMS approach in the target classification problem, and compare its performance to the discrete max-sum algorithm. In so doing, we show that HCMS outperforms discrete max-sum by up to 30%.
3. We further show that the improvements in solution quality that the HCMS algorithm achieves over discrete max-sum come with neither significant increases in running time nor communication cost.

The remainder of this chapter is structured as follows. In Section 5.1 we formally describe coordination problem common to both wide area surveillance problems. In Section 5.2 we develop the CPLF-MS algorithm for piecewise linear functions. In Section 5.3 we describe the event detection problem for energy constrained agents, whose interactions can be expressed as piecewise linear functions, and evaluate the CPLF-MS algorithm. Then, in Section 5.4 we give a formal description of HCMS, and statement and proof

[4]This is joint work with Tom Voice, who laid the foundation for the algorithm and proved its theoretical properties. I carried out the implementation and empirical analysis, which led to further improvements to the algorithm in collaboration with Tom.

of the theoretical results. In Section 5.5, we empirically evaluate its performance in the target classification problem. Finally, in Section 5.6, we summarise the contributions made in this chapter and evaluate the developed algorithms in terms of the design requirements defined in Chapter 1.

## 5.1 Problem Description

We now formally describe the decentralised coordination problem that we address in this chapter. This problem extends the problem formulated in Chapter 3 to fixed agents with continuous control variables. For these settings, we assume that the observation value received by the team of agents can be decomposed into a set of agent-dependent factors, such that the max-sum algorithm can be applied. Later in this chapter, we show how a good decomposition can be derived for the two wide-area surveillance scenarios.

The setup of these scenarios is similar to the one we studied in the previous chapter. We have a set of $M$ agents $\mathbf{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_M\}$ deployed on the $\mathbb{R}^2$ plane. Each agent $\mathcal{A}_i$ has an observation area $\bigcirc_i$ within which it can observe events. Furthermore, each agent has a single[5] continuous action variable $p_i$ whose domain $\mathcal{D}_i$ is a closed and bounded interval in $\mathbb{R}$. By changing the value of this variable, the agent controls the set of observations it makes. For example, by changing its viewing orientation in Figure 5.1, agent $\mathcal{A}_2$ can choose to observe target $t_1$ instead of $t_2$ (or none at all).

Since the observation areas of agents can overlap, it is possible that multiple agents observe the same features of their environment. Since we assume that these observations are not independent (see Chapter 3), the value of these observations is sub-additive (through the property of submodularity of the observation value function $f$). As a result, the observation value received by agent $\mathcal{A}_i$ depends on the observations made by $\mathcal{A}_i$, and thus $p_i$, but also on the observations made by the agents whose observation areas overlap with its own. Using the familiar notation of the max-sum algorithm (Section 2.6.1), we express this observation value as a utility function $U_i(\mathbf{p}_i)$, where $\mathbf{p}_i$ is the vector of variables that influence its utility (and thus we have that $p_i \in \mathbf{p}_i$). Since these variables are continuous, utility functions $U_i(\mathbf{p}_i)$ are multivariate continuous functions.

As in Chapter 3, the key challenge faced by the agents to maximise observation value as a team. Using the utility functions and action variables mentioned above, this challenge can be translated into a welfare optimisation problem; the agents need to compute the joint action that maximises:

$$\mathbf{a}^* = \arg\max_{\mathbf{p}} \sum_{i=1}^{M} U_i(\mathbf{p}_i) \tag{5.1}$$

[5]Note both algorithms we develop in this chapter allow multiple variables per agent, with possibly different domains, but for ease of exposition we present just the single variable case here.

In order to enforce a truly decentralised solution, we assume that each agent only has knowledge of, and can directly communicate with, the few neighbouring agents that influence its own utility directly.

The point of departure for developing the two new algorithms for continuous coordination problems is the max-sum algorithm, which we discussed in Section 2.6.1. For convenience, we repeat the equations for computing the messages that are exchanged between functions and variables below:

- **From variable $p_i$ to function $U_j$:**

$$q_{i\to j}(p_i) = \alpha_{ij} + \sum_{k\in\mathcal{M}_i\setminus j} r_{k\to i}(p_i) \tag{5.2}$$

- **From function $U_j$ to variable $p_i$:**

$$r_{j\to i}(p_i) = \arg\max_{\mathbf{p}_j\setminus p_i} \left[ U_j(\mathbf{p}_j) + \sum_{k\in\mathcal{N}_j\setminus i} q_{k\to j}(p_k) \right] \tag{5.3}$$

Note these equations apply equally well to both continuously valued and discrete variables. However, the two core mathematical operations, summation and marginal maximisation, required in Equations 5.2 and 5.3 are much more readily implemented in the case of discrete variables. In Sections 5.2 and 5.4, where we present the two new algorithms, we show how these operations can be defined for piecewise linear and non-linear utility functions.

## 5.2 CPLF-MS: Max-Sum for Piecewise Linear Functions

CPLF-MS is the first extension to the max-sum algorithm for solving coordination problems where the agents' utility functions are expressible as a multivariate continuous piecewise linear functions (CPLF). In order to apply the max-sum algorithm to these settings, we need to be able to perform the mathematical operations—summation and marginal maximisation—on these functions. Under the restriction that the utility functions are CPLFs, the two aforementioned operations have an intuitive geometric interpretation that makes it possible to define and manipulate them using standard techniques from computational geometry, hence allowing the continuous versions of the operations required by max-sum to be performed. More specifically, in this section we show how to:

1. Represent each agent's utility function as a CPLF (Section 5.2.1).

2. Perform the summation of two CPLFs (Section 5.2.2). This is required in order to perform the addition in Equation 5.3:

$$U_j(\mathbf{p}_j) + \sum_{k \in \mathcal{N}_j \setminus i} q_{k \to j}(p_k)$$

where both operands are now CPLFs.

3. Calculate the marginal maximisation of a CPLF with respect to a single variable (Section 5.2.3). This operation is needed in order to find the maximum of a CPLF with respect to a set of variables $\mathbf{p}_j \setminus p_i$ in Equation 5.3:

$$\max_{\mathbf{p}_j \setminus p_i} \left[ U_j(\mathbf{p}_j) + \sum_{k \in \mathcal{N}_j \setminus i} q_{k \to j}(p_k) \right]$$

4. Use the operators defined in Sections 5.2.3 and 5.2.2 to instantiate the CPLF-MS algorithm (Section 5.2.4).

### 5.2.1 Representing CPLFs with Simplexes

A CPLF is a function whose domain can be partitioned into a set of convex polytopes,[6] such that it is linear on each of these polytopes. For example, for one variable, a CPLF is a function that can be represented with a finite number of line segments, and for two variables a CPLF can be represented by a finite number of two-dimensional polygons. Figure 5.2 shows an example of the latter. The domain of this function is partitioned into 14 triangles (shown on the $(p_1, p_2)$ plane) such that the function is indeed linear on each of them.

In our formalism, we use $n$-dimensional simplexes, or $n$-simplexes, to partition the domain of an $n$-ary CPLF. The reason for this is that an $n$-simplex is the simplest $n$-dimensional polytope and it is therefore easy to manipulate. An $n$-simplex is constructed by taking the convex hull of a set of $n+1$ affinely independent points $\{\mathbf{x}_1, \ldots, \mathbf{x}_{n+1}\} \in \mathbb{R}^m$ $(m \geq n)$, and is denoted by $\Delta^n$. We will omit the superscript $n$ when the dimensionality is clear from the context. The set of points enclosed by a simplex is given by:

$$\Delta^n = \left\{ \mathbf{x} \in \mathbb{R}^m \mid \sum_{i=1}^{n} a_i \mathbf{x}_i = \mathbf{x}, \sum_i a_i = 1, \forall i : a_i \geq 0 \right\} \tag{5.4}$$

Now, an $n$-ary CPLF $f : \mathcal{D} \to \mathbb{R}$ is defined by a set $\{\Delta_1, \ldots, \Delta_m\}$ of $n$-simplexes in $\mathbb{R}^{n+1}$. The functions' domain $\mathcal{D}$ is the Cartesian product of the domains of variables $(p_1, \ldots, p_n)$,[7] i.e. $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_n$. Since each $\mathcal{D}_i$ is a closed interval in $\mathbb{R}$, $\mathcal{D}$ is

[6] A convex polytope is a multi-dimensional generalisation of the two-dimensional convex polygon. In $n$ dimensions, it is a convex hull of at least $n + 1$ points.

[7] In what follows, $(p_1, \ldots, p_n)$ and $\mathbf{p}$ are used interchangeably.

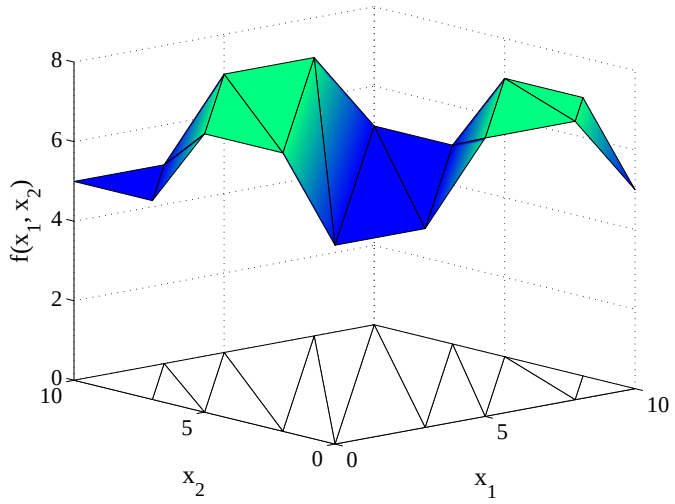
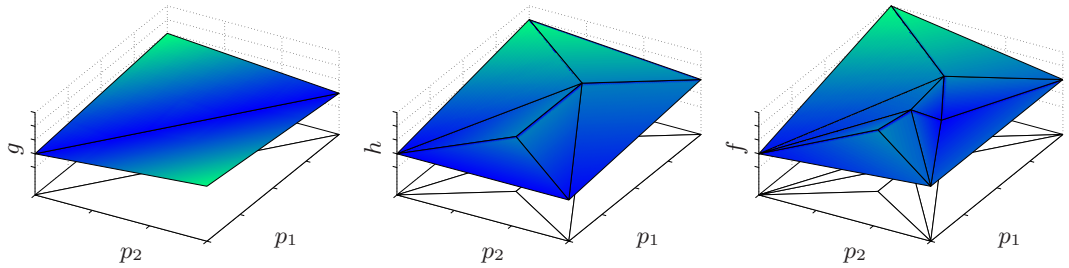



FIGURE 5.2: An example of a bivariate CPLF.




(a) Function $g$ (b) Function $h$ (c) Function $f = g \oplus h$

FIGURE 5.3: Three example CPLFs

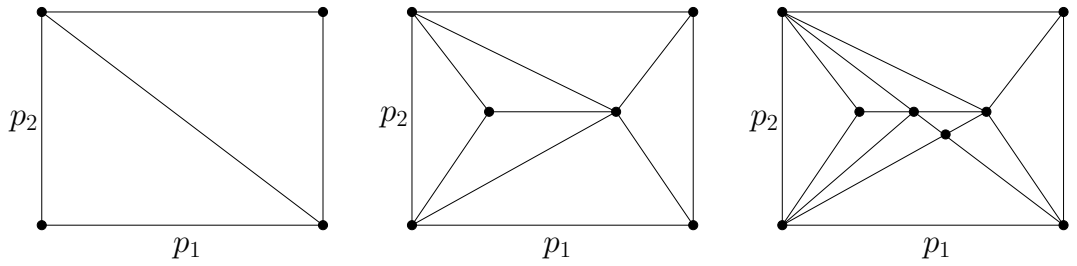
an interval in $\mathbb{R}^n$, or an $n$-cube. From the definition of a CPLF, we require that the projection of the simplexes that define $f$ is a partition $P_{\mathcal{D}}$ of $\mathcal{D}$. (For instance, in Figure 5.2 the projection of the simplexes onto the $(p_1, p_2)$ plane covers this plane without overlapping.) More formally,

$$P_{\mathcal{D}} = \left\{ \widetilde{\Delta}_i \mid 1 \leq i \leq m, \bigcup_i \widetilde{\Delta}_i = \mathcal{D}, \widetilde{\Delta}_i \bigcap \widetilde{\Delta}_j = \emptyset, 1 \leq i < j \leq m \right\} \tag{5.5}$$

where $\widetilde{\Delta}_i$ is the projection of $\Delta_i$ onto the $\mathbf{p}$ hyperplane.

Furthermore, in order to guarantee that $f$ is continuous, we must ensure that simplexes whose projection onto the $\mathbf{p}$ hyperplane share the same coordinates in $\mathbb{R}^n$, also share the same coordinates in $\mathbb{R}^{n+1}$.

Given this representation of a utility function, we can now derive exact algorithmic solutions for computing the two fundamental operations required for implementing our continuous max-sum algorithm.




(a) Domain partition of function $g$ (b) Domain partition of function $h$ (c) Domain partition of function $f = g \oplus h$

FIGURE 5.4: Domain partitions of the functions in Figure 5.3.

### 5.2.2 Summation of Two CPLFs

In order to perform the summation of two CPLFs $g$ and $h$ with identical domains, we need to compute the simplexes that make up function $f$ such that $\forall \mathbf{x} \in \mathcal{D} : f(\mathbf{x}) = g(\mathbf{x}) + h(\mathbf{x})$ holds. We denote the operator that adds two CPLFs as $\oplus$. This operator works in in two steps. First, it computes the domain partition $P_f$ of $f$, such that $P_f$ contains a corner[8] at every corner in $g$ and $h$. Second, it computes the values of $f$ at each corner point $\mathbf{x}$ of the simplexes that partition $f$. The latter step is trivial; for each corner point $\mathbf{x}$, evaluate $g(\mathbf{x}) + h(\mathbf{x})$. However, the former step is a little more involved, since computing the domain partition of $f$ involves overlaying or *merging* the domain partitions $P_g$ and $P_h$ in order to determine where the sum of $g$ and $h$ might have a corner.

The following example illustrates this operation for two two-dimensional functions.

**Example 5.1.** *Consider the functions $g$ and $h$ in Figures 5.3(a) and 5.3(b). Their domain partitions are shown in Figures 5.4(a) and 5.4(b). Function $f$ is the sum of these two functions, shown in 5.3(c). Note that the partition of $f$ shown in Figure 5.4(c) indeed has corners at every location where function $g$ and $h$ have corners, including two new ones.*

Algorithm 5 shows the necessary computation for finding the domain partition of $f$, which proceeds in two main steps:

1. Copy partition $P_g$ to the variable $P_f$ that contains the result while it is constructed (line 1).
2. Compute the intersection of every simplex in $P_h$ with every simplex of $P_f$:
   (a) Add the vertexes of all simplexes in $P_h$ to $P_f$ (lines 2 to 6).
   (b) Add the edges of the simplexes in $P_h$ to $P_f$ (lines 7 to 13). These edges are the corners of $h$, and therefore need to be present in $P_f$.

**Algorithm 5** An algorithm for merging two partitions

***Require:*** *Partitions* $P_g$ *and* $P_h$
***Ensure:*** *Partition* $P_f = P_g \oplus P_h$
1: $P_f \leftarrow P_g$
2: ***for all*** $\Delta \in P_h$ ***do***
3: ***for all*** $\mathbf{x} \in \{\mathbf{x}_1, \ldots, \mathbf{x}_{n+1}\}$ *that define* $\Delta$ ***do***
4: $P_f \leftarrow \mathcal{S}(P_f, \mathbf{x})$
5: ***end for***
6: ***end for***
7: ***for all*** $\Delta_h \in P_h$ ***do***
8: ***for all*** $\Delta_f \in P_f$ ***do***
9: ***for all*** *intersections* $\mathbf{x}$ *of the edges of* $\Delta_h$ *with the* $(n-1)$*-faces of* $\Delta_f$ ***do***
10: $P_f \leftarrow \mathcal{S}(P_f, \mathbf{x})$
11: ***end for***
12: ***end for***
13: ***end for***
14: ***return*** $P_f$

To complete the specification of Algorithm 5, we need to define the split operator $\mathcal{S}$ used in finding the intersections between two simplexes (lines 4 and 10). Specifically, the split operator $\mathcal{S}$ partitions a simplex $\Delta^n$ around a point $\mathbf{x}$: $\mathcal{S}(\Delta^n, \mathbf{x}) = \{\Delta^n_1, \ldots \Delta^n_m\}$. Thus, each $\Delta^n_i \in \mathcal{S}(\Delta^n, \mathbf{x})$ is obtained by creating a simplex with vertexes $\{\mathbf{x}, \mathbf{x}_1, \ldots, \mathbf{x}_{n+1}\} \setminus \{\mathbf{x}_i\}$. Depending on the location of $\mathbf{x}$ in $\Delta^n$, the split operator creates a different number of simplexes. In more detail, depending on the complexity of the face[9] of $\Delta^n$ on which $\mathbf{x}$ lies, $\mathcal{S}$ splits $\Delta^n$ into at least 1, and at most $n$ simplexes. Figures 5.5(b) and 5.5(c) show how the 2-simplex in 5.5(a) is split on points on a 2-face (body) and a 1-face (edge) respectively. Note that, in the latter case, the simplex is split in two, since vertices $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}\}$ are not affinely independent, and therefore do not form a simplex. Splitting on a 0-face (or vertex) does not split the simplex, neither does splitting on a point outside the simplex. To avoid cluttering the notation in Algorithm 5, we denote the operation of splitting all simplexes in a partition $P$ on a point as $\mathcal{S}(P, \mathbf{x})$, which is shorthand for $\cup_{\Delta \in P} \mathcal{S}(\Delta, \mathbf{x})$.

### 5.2.3 Marginal Maximisation of a CPLF

Marginal maximisation is the second operator that is needed in max-sum. It takes as input a function $y = f(p_1, \ldots, p_n)$ and a variable $p_i$, and computes a single-dimensional CPLF $f(p_i) = \max_{\mathbf{p} \setminus p_i} f(p_1, \ldots, p_n)$.

The computation of the marginal maximisation of a CPLF proceeds in two steps:

[8] A corner is the location at which two simplexes meet at an angle.

[9] The faces of a n-simplex are (n-1)-simplexes that make up its boundaries. The *complexity* of a face of an $n$-simplex is its dimensionality, which ranges from 1 to $n$. A face of complexity $i$ is called an $i$-face. A 0-face is a vertex of the simplex, a 1-face is an edge, a 2-face is a triangle, etc. The $n$-face of the simplex is the simplex itself, which is also referred to as the *body*.

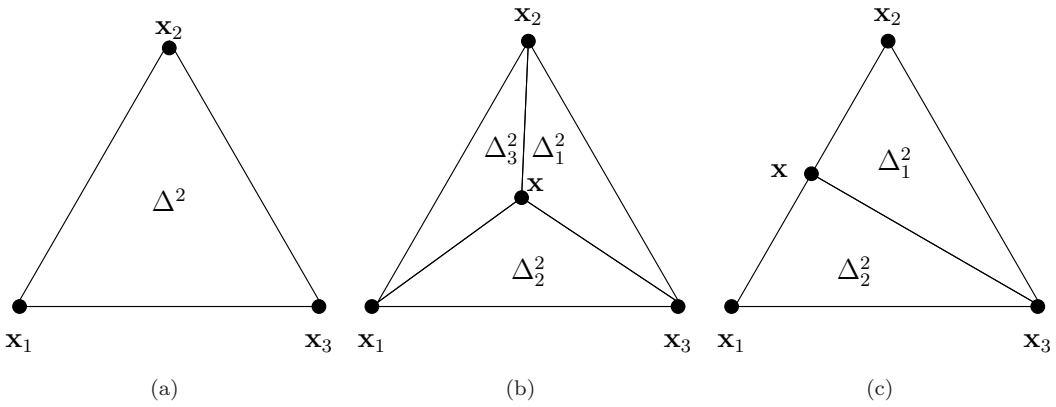



FIGURE 5.5: (a) A 2-simplex. (b) Splitting a 2-simplex on point $\mathbf{x}$ on a 2-face. (c) Splitting a 2-simplex on point $\mathbf{x}$ on a 1-face (edge)

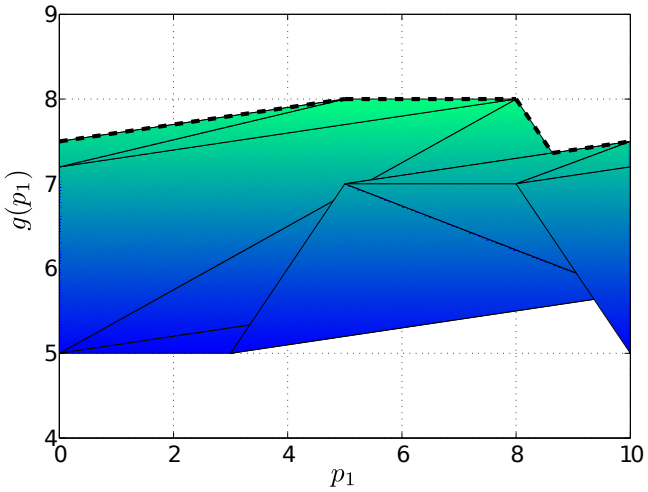



FIGURE 5.6: A CPLF $y = g(p_1, p_2)$ projected onto the $(p_1, y)$ plane. The dotted line indicates the upper envelope of these simplexes, and equals $g(p_1) = \max_{p_2} g(p_1, p_2)$.

1. Project all simplexes that define $f$ onto the $(p_i, y)$ plane. An n-simplex $\Delta$ is projected by projecting each of its $m = \binom{n}{2}$ edges to obtain a set of line segments $S_\Delta = \{s_1, \ldots, s_m\}$.

2. Extract the *upper envelope* of the line segments in $S_\Delta$. The upper envelope is a function $\hat{U}_S$ of the set $S$ of all projected line segments of all simplexes that make up $f$ is then a function:

$$\hat{U}_S(p_i) = \max\{s(p_i) \mid s \in S \wedge p_i \in [s_s, s_e]\}$$

where $[s_s, s_e]$ is the closed interval on which line segment $s$ is defined. The upper envelope of a set of $n$ line segments can be computed in $O(n \log n)$ operations (Hershberger 1989).

The following example demonstrates this operation for a two dimensional function.

**Example 5.2.** *Consider the function $y = g(p_1, p_2) = f(p_1, p_2) + 0.1p_2$, where $f(\cdot, \cdot)$ is the function in Figure 5.2. This function $g(p_1, p_2)$ is the sum of the function $f$ and a function $q_{1\to f}(p_2) = 0.1p_2$, received from variable $p_2$. The dotted line in Figure 5.6 shows the upper envelope of $g$'s simplexes projected onto the $(p_1, y)$ plane, which is the result of performing the marginal maximisation operator on this function with respect to $p_1$. Moreover, it is the message sent from function $f$ to variable $p_2$ computed by Equation 5.3.*

### 5.2.4 Instantiating the CPLF-MS Algorithm

Now that we have defined the utility functions as CPLFs and the two required mathematical operations, we can instantiate the CPLF-MS algorithm for continuous variables, by defining the processes through which messages between the variables and functions are computed:

- From variable to function (Equation 5.2). Since the messages $r_{j\to i}(p_i)$ are real-valued functions of a single continuous variable $p_i$, the computation of $q_{i\to j}$ involves summing over single-dimensional CPLFs using the $\oplus$ operator of Section 5.2.2. The addition of scalar $a_{ij}$ is trivial.
- From function to variable (Equation 5.3). The computation of the message $r_{j\to i}(p_i)$ proceeds in two steps. First, the expression between the brackets is evaluated. The first term in this expression is the utility of agent $\mathcal{A}_j$, which is a CPLF. The second term is the sum of multiple single-dimensional CPLFs of *different* variables, which is a multi-dimensional CPLF. So, to evaluate this expression, we sum the first and second term using the $\oplus$ operator, which, again, results in a CPLF. Second, we use the marginal maximisation operator on this CPLF to obtain the message as required.

Now that we have performed the necessary steps to instantiate the continuous max-sum algorithm, we should note that there is a downside to using simplexes to define CPLFs. While they are simple to manipulate, the summation of incoming messages with the utility function in Equation 5.3 often yields functions with a large number of simplexes. Moreover, the functions are not represented as compactly as possible. Turning to Figure 5.3 we see an example of this. The addition of two fairly simple functions results in a function with 10 simplexes. Furthermore, Figure 5.2, requires 14 simplexes, while 8 *polygons* would suffice. Increasing the dimensionality of the function (i.e. making it a function of more parameters) only exacerbates this problem. We discuss how this influences the performance of the algorithm in the next section, where we compare the performance of CPLF-MS and the discrete max-sum algorithm in the event detection scenario.

## 5.3 The Event Detection Scenario

In the event detection scenario used to benchmark the CPLF-MS algorithm, a team of fixed information gathering agents is randomly deployed within some area to detect events (e.g. vehicle and pedestrian activity in an urban setting). We assume that these agents are able to harvest energy from the environment (e.g. using a photovoltaic cell or vibration-harvesting microgenerators), but at a rate that is insufficient to allow them to be powered continually. Thus, at any time an agent can be in one of two states: either sensing or sleeping. In the former state, the agent consumes energy at a constant rate, and is able to interact with the surrounding environment (e.g. it can detect events within its observation area and communicate with other agents). In the latter state, the agent cannot interact with the environment but it consumes negligible energy. To maintain energy-neutral operation (Kansal et al. 2007), and thus exhibit an indefinite lifetime, agents adopt a repeated schedule of length $L$, during which each agent can be active for only a given time $l_i < L$. This amount of time depends on specific characteristics of the environment surrounding the agent, and the means by which energy is harvested. For example, if an agent is equipped with solar panels to harvest energy, agents that are in shaded regions will have shorter duty cycles compared to those with a greater exposure to sunlight.

In the remainder of this section, we instantiate the problem defined in Section 5.1, by decomposing the observation value obtained by the agents into a set of utility functions—one for each agent.

As mentioned in Section 5.6, the observation areas of multiple agents will typically overlap. However, just a single agent is required to be active in order to detect an event. Thus, there is no gain for the system in having more than one agent actively monitoring the same region (i.e. we have a strongly submodular utility function), and hence, to maximise the probability of detecting events while maintaining energy neutral operations, agents whose observation areas overlap should coordinate the activation times of their duty cycles. Therefore, in this setting, the continuously valued action variable $p_i$ represents the time at which agent $\mathcal{A}_i$ will start sensing, while the domain over which this variable can take values is the interval $[0, L]$. Once the agents have decided on the value of this parameter, they will repeat this schedule indefinitely.

In order to apply the max-sum algorithm, in either its continuous or discrete forms, we now show how to instantiate the general coordination problem defined in Section 5.1 for this scenario, by defining the agents' utility functions. These utility functions are expressed in terms of observation value, which, in turn, are expressed in terms of the probability of detecting events. Since these events are randomly distributed within the area, this probability is proportional to the area that is observed by the active agents. As before, we denote the area that agent $\mathcal{A}_i$ observes as $\bigcirc_i$. Furthermore, $A_{\{i,\mathbf{k}\}}$ is the area that is *only* observed by agent $\mathcal{A}_i$ and the agents $\mathcal{A}_k$ with $k \in \mathbf{k}$. For example, with

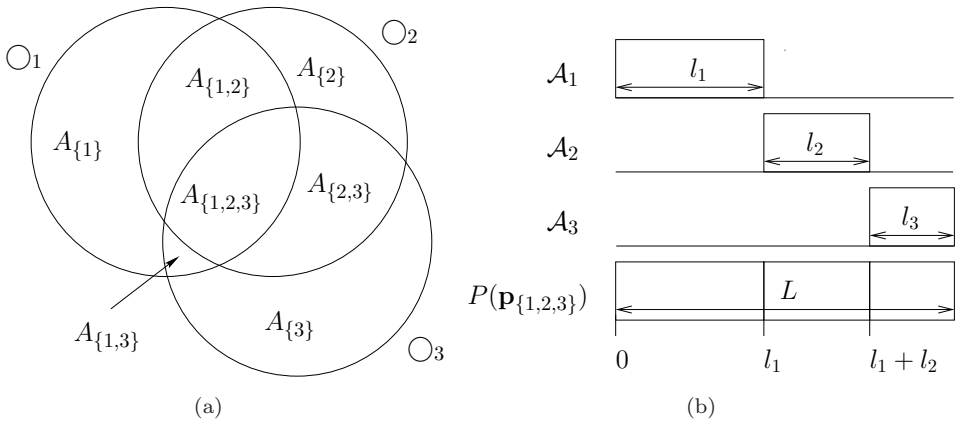



FIGURE 5.7: (a) Example coordination problem in which three agents, $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, have sensing fields that overlap (b) An optimal solution

respect to Figure 5.7(a), which shows the three agents whose sensing areas overlap, the area $A_{\{1,2\}}$ is the area that is sensed only by agents $\mathcal{A}_1$ and $\mathcal{A}_2$. Note that we represent the entire observation areas of agents $\mathcal{A}_1$ and $\mathcal{A}_2$ as $\bigcirc_1$ and $\bigcirc_2$, and thus, area $A_{\{1,2\}}$ is different from $\bigcirc_1 \cap \bigcirc_2$, since the latter would also include subarea $\bigcirc_1 \cap \bigcirc_2 \cap \bigcirc_3$. In general, we have:

$$A_{\{i\}\cup\mathbf{k}} = \bigcap_{j\in(\{i\}\cup\mathbf{k})} \bigcirc_j \setminus \bigcup_{l\notin(\{i\}\cup\mathbf{k})} \bigcirc_l \tag{5.6}$$

The utility of agent $\mathcal{A}_i$ is then simply given by the weighted sum of the probability of detecting an event in any of the subarea it observes:

$$U_i(\mathbf{x}_i) = \sum_{\mathbf{k}\subseteq\{1,\ldots,M\}} \frac{A_{\{i,\mathbf{k}\}}}{|\{i\}\cup\mathbf{k}|} \times P(\mathbf{p}_{\{i\}\cup\mathbf{k}}) \tag{5.7}$$

where $\mathbf{p}_{\{i\}\cup\mathbf{k}}$ is the vector of variables corresponding to agents $\{\mathcal{A}_i\} \cup \{\mathcal{A}_k\}_{k\in\mathbf{k}}$, which represents the combined activation schedule of these agents, and $P(\mathbf{p}_{\{i\}\cup\mathbf{k}})$ is the probability of detecting an event per unit area given this combined activation schedules. To ensure the functions are piecewise linear—and facilitate the evaluation of the CPLF-MS algorithm—we assume this probability is given by the fraction of the time during which at least one agent is actively sensing during the interval of length $L$, and hence, we are assuming that events are instantaneous in time and have no duration.

Note that in Equation 5.7 we divide each subarea by the number of agents that can sense it to avoid double-counting areas that are represented by multiple agents. In addition, when the set $\mathbf{k}$ is empty we consider the area covered only by the single agent. For example, the utility of agent $\mathcal{A}_2$ shown in Figure 5.7(a), is calculated by considering the areas $A_{\{2\}}$, $A_{\{1,2\}}$, $A_{\{2,3\}}$ and $A_{\{1,2,3\}}$.

Consider the following example that illustrates this formalisation.

**Example 5.3.** *Suppose that agents are deployed as in Figure 5.7(a), and suppose that to maintain energy neutral operations, the three agents can be active for $l_1$, $l_2$, and $l_3$ time units out of $L$ (with $l_1 + l_2 + l_3 = L$). In this case, an optimal solution is the one reported in Figure 5.7(b) where $p_1 = 0$, $p_2 = l_1$ and $p_3 = l_1 + l_2$. Note that, in this case, there is an infinite number of optimal solutions, which can be generated by shifting the activation times of all agents by an equal amount. In an optimal solution, $\mathcal{A}_1$ receives an observation value of:*

$$A_{\{1\}} l_1 + \frac{A_{\{1,2\}}}{2}(l_1 + l_2) + \frac{A_{\{1,3\}}}{2}(l_1 + l_3) + \frac{A_{\{1,2,3\}}}{3} L$$

*Similar results hold for agents $\mathcal{A}_2$ and $\mathcal{A}_3$.*

*This scenario also illustrates the difference between discrete max-sum and a CPLF-MS. Notice that, while continuous max-sum is able to assign any value in the interval $[0, L]$ to the agents' variables, discrete max-sum will be able to find an optimal solution only if the chosen discretisation includes the points $l_1$ and $l_1 + l_2$. However, as previously mentioned, the agents' duty cycles depend on where the agents are deployed and on the type of environment, and thus they are not known before-hand. Hence, it is not possible to always choose a discretisation that includes the optimal solution and thus discrete max-sum will yield suboptimal solutions.*

Given these utility functions and the agents' action variables, we can now construct a factor graph for this problem, by connecting the utility function of agent $\mathcal{A}_i$ in Equation 5.7 to the variables of all agents whose observation field overlaps with that of $\mathcal{A}_i$. We can now directly apply the discrete max-sum algorithm and CPLF-MS to this factor graph, and compare their performance.

### 5.3.1 Empirical Evaluation

To empirically evaluate the performance of the CPLF-MS algorithm, we benchmark it against the discrete max-sum algorithm. Before we do this, however, we first need to express the utility function in 5.7 as a CPLF. For an area where two agents overlap, this function is shown in Figure 5.2. The two agents in question can operate $l_1 = 2$, and $l_2 = 5$ out of $L = 10$ time units. The function exhibits a minimum plateau when the active sensing periods of both agents completely overlap in time (e.g. when $p_1 = p_2 = 0$), and a maximum plateau when there is no overlap in the agents' schedules (e.g. when $p_1 = 0$ and $p_2 \in [2, 5]$). Notice that if we fix $p_1 = 0$ while increasing $p_2$, the utility received by both agents increases as well. This is because the length of the interval in which at least one of the two agents is sensing, is increasing. When $p_2 = 2$, the function assumes the maximum value and then remains constant until $p_2 = 5$. From $p_2 = 5$,

the function starts decreasing. This is because the interval of $\mathcal{A}_2$ wraps around, and it starts sensing during the intervals $[p_2, 10]$ and $[0, 10 - p_2]$, the latter of which overlaps with the sensing interval of $\mathcal{A}_1$, thus decreasing the agents' utility.

As mentioned at the end of Section 5.2.4, the CPLF-MS algorithm has a propensity for generating a large number of simplexes during the course of its operation. In fact, while attempting to run this algorithm on problem instances where the observation areas of more than two agents overlap, thus creating components of three or more parameters in the agents' utility functions (Equation 5.7), we found that the addition in Equation 5.3 often produced functions with hundreds of simplexes. In part, this is attributable to numerical instabilities inherent in our rudimentary implementation of this algorithm, but the main cause of this problem lies in the use of simplexes as discussed above.

In light of this, to facilitate comparison between CPLF-MS and the discrete max-sum algorithm, we ignored areas where more than two agents overlap, which is equivalent to only considering pairwise interactions between agents. This is actually a very common approach in the DCOP literature (Modi et al. 2005, Mailler & Lesser 2008, Petcu & Faltings 2005), and one that reduces the computational complexity of the coordination, while still providing good solutions in this particular scenario.

Given this configuration of our algorithm, we now empirically evaluate its performance.

### 5.3.2 Experimental Results

We benchmarked the CPLF-MS algorithm against two algorithms:

**Discrete Max-Sum** The discrete max-sum algorithm is applied to the same factor graph as CPLF-MS, but the domain $[0, L]$ of the action variables is artificially discretised into $d$ discrete values.

**Centralised Simulated Annealing** This is a centralised algorithm for solving continuous optimisation problems which often yields optimal results (Granville et al. 1994). We include this algorithm to provide an upper bound on achievable performance and to normalise the solution quality of CPLF-MS and discrete max-sum.

We measured the quality of the solution and the communication overhead on deployments of 10 agents, which are randomly scattered across a unit square. These agents have a circular shaped observation area with a radius of 0.2. The agents' duty cycles $l_i$ are drawn from a uniform distribution over $[0.3, 0.6]$. These values were chosen after initial calibration showed that they produced particularly challenging coordination problems.

We ran both CPLF-MS and the discrete max-sum algorithm for 20 iterations. Each experiment consisted of a single run of our CPLF-MS, and multiple runs of the discrete max-sum algorithm with increasingly fine discretisations. Figure 5.8 shows the aggregated results of 100 runs, where the solution quality is normalised against the solution quality of the centralised simulated annealing algorithm.

Specifically, Figure 5.8(a) reports the quality of the final solution (e.g., after the 20 iterations), while Figure 5.8(b) reports the average quality of the solutions obtained after each iteration. The latter metric incorporates information on how the algorithms behave over time; the quicker the algorithms converge towards better solutions, the higher the average.

Figure 5.8(a) shows that the final solutions produced by continuous max-sum are better than those produced by the discrete version. In particular, CPLF-MS exhibits up to a 10% increase in the solution quality for low discretisation levels.

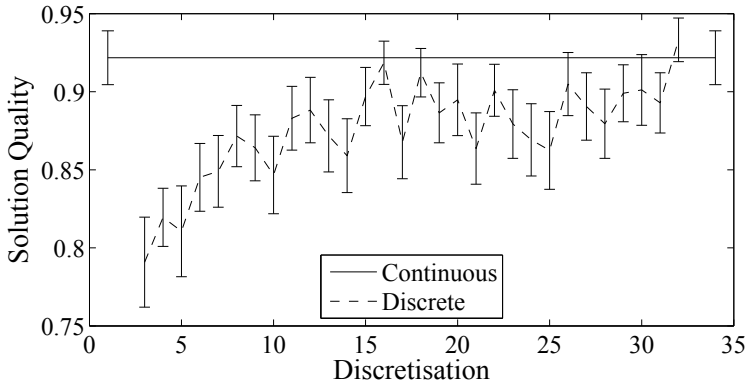
Moreover, Figure 5.8(b) shows that, when considering the average quality of solution over multiple iterations, this difference is more pronounced, thus showing that CPLF-MS is able to reach good, stable solutions quicker than the discrete version. This increased performance for the average solution can partially be explained by the faster convergence speed of CPLF-MS: since it takes discrete max-sum longer to converge to a good solution, its solution quality averaged over all 20 iterations is lower than that of CPLF-MS.

In terms of total message size, we can conclude from Figure 5.9 that, as expected, the communication overhead of discrete max-sum increases proportionally with the level of discretisation. Most importantly, the CPLF-MS achieves a better solution quality over the entire range of discretisations, even when the message size of the discrete max-sum algorithm is greater than that of CPLF-MS. Thus, the latter generates better solutions, and also requires less communication overhead.

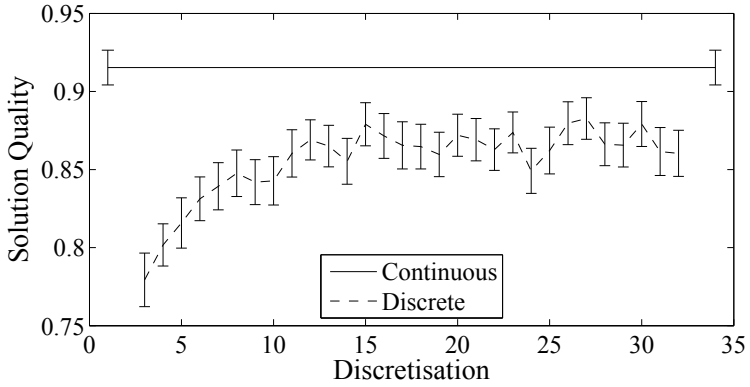
This concludes the empirical evaluation of the CPLF-MS algorithm for continuous control parameters. While we have shown it outperforms the discrete max-sum algorithm, its use is limited to settings where the interactions of agents can be represented by piecewise linear utility functions. Moreover, the complexity of the representation of the utility functions tends to scale unfavourably with the number of neighbouring agents. These drawbacks have encouraged us to develop the HCMS algorithm for non-linear functions, which does not attempt to derive an exact implementation of the mathematical operations required by max-sum, and by so doing, avoids these problems.

## 5.4 HCMS: Max-Sum for Non-Linear Utility Functions

HCMS, hybrid continuous max-sum, is the second extension to the max-sum algorithm. Recall from Section 2.6.1 that the standard max-sum algorithm requires variables $p_i$




(a) Solution quality after 20 iterations




(b) Averaged solution quality over 20 iterations

FIGURE 5.8: Solution quality as a fraction of the solution quality computed by simulated annealing. Error bars are the standard error in the mean.

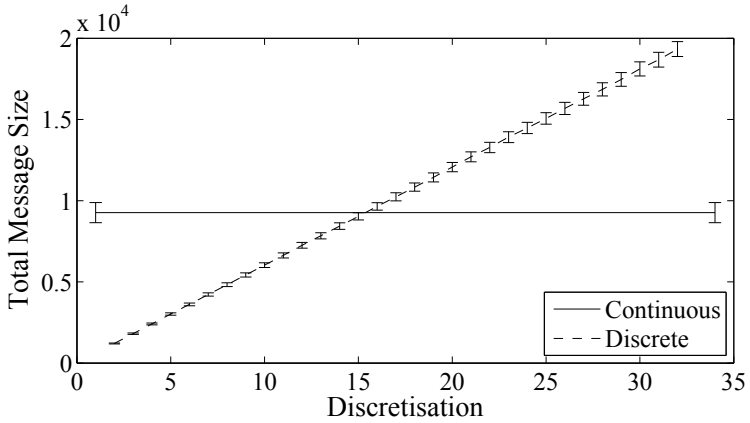



FIGURE 5.9: Total number of values exchanged between the agents. Error bars are the standard error in the mean.

with *discrete* domains $\mathcal{D}_i = \{a_i^1, a_i^1, \ldots, a_i^{k_i}\}$. As mentioned earlier, should we wish to use max-sum to solve optimisation problems with continuous variables, we can proceed by discretising the domain of each variable and run max-sum on this modified problem. In this case, the max-sum algorithm finds an approximately optimal set of states within this discretisation. The key principle behind the HCMS algorithm is that it adjusts this discretisation to seek better quality solutions at each iteration of the max-sum algorithm. This adjustment is guided by the gradient of the global objective function (i.e. the sum of utility functions in Equation 2.9) using non-linear optimisation techniques.

In more detail, the HCMS algorithm involves implementing the same message passing as described in Equations 5.2 and 5.3, using the current discretisations of the variables' domains. In addition to this, it also sends updates about these domains, as well as information about the gradient of the global function (Equation 5.1). In more detail, firstly, each variable $p_i$ must communicate to all functions $U_j$, for $j \in \mathcal{M}_i$, the values in its current domain $\mathcal{D}_i$. Secondly, each utility function $U_j$ communicates to each variable $p_i$ for $i \in \mathcal{N}_j$ either $f^1_{j\to i}(\cdot)$, or both $f^1_{j\to i}(\cdot)$ and $f^2_{j\to i}(\cdot)$, where for $n = 1, 2$, $f^n_{j\to i}(p_i)$ is given by:

$$\frac{d^n}{dp_j^n} \arg\max_{\mathbf{p}_j \setminus p_i} \left[ U_j(\mathbf{p}_j) + \sum_{k \in \mathcal{N}_j \setminus i} q_{k\to j}(p_k) \right] \tag{5.8}$$

As described above, the key difference between the HCMS and the discrete max-sum algorithm, is that each variable $p_i$ evolves its domain in order to find better quality solutions. To do so, variable $p_i$ employs continuous non-linear optimisation techniques, which proceed as if maximising an objective function which equals the sum of the messages received from functions $\sum_{j\in\mathcal{M}_i} r_{j\to i}(p_i)$ with $n^{\text{th}}$ gradient $\sum_{j\in\mathcal{M}_i} f^n_{j\to i}(a_i)$ at each point in its domain $a_i \in \mathcal{D}_i$. The motivation for this is that, as a result of the discrete max-sum message passing process, for each variable $p_i$, for all $a_i \in \mathcal{D}_i$, the received values of $\sum_{j\in\mathcal{M}_i} r_{j\to i}(a_i)$ can be used as an approximation to the marginal function $\tilde{U}_i(a_i)$ evaluated at $a_i$ (Equation 2.10). Furthermore, $\sum_{j\in\mathcal{M}_i} f^n_{j\to i}(a_i)$, can be used as an approximation of the $n^{\text{th}}$ gradient of the marginal function:

$$\frac{d^n \tilde{U}_i}{dp_i^n}$$

evaluated at $a_i$, which is used to update $a_i$'s value in order to maximise the global objective function, as we show next.

### 5.4.1 Continuous Non-linear Optimisation

Since each variable attempts to optimise its marginal function using a series of approximations, it is important that robust optimisation methods are chosen. We choose gradient methods, which are robust to errors and can be implemented in a highly scalable, asynchronous, decentralised fashion using only local information (Avriel 1999). This leads to the intuition that the different state updates of the variables will not interact in unpredictable or harmful ways.

With these gradient methods, each variable updates its domain after each iteration of the HCMS message passing process, by adding $\Delta a_i^z$ to $a_i^z$ for each $z = 1, \ldots, k_i$ (recall that $k_i$ is the number of elements in $\mathcal{D}_i$), where

$$\Delta a_i^z = \kappa_i(z) \sum_{j \in \mathcal{M}_i} f^1_{j \to i}(a_i^z)$$

To complete the HCMS algorithm, scaling factor $\kappa_i(z)$ has to be chosen. Here, we consider two schemes for setting this parameter. Firstly, we consider a straightforward gradient method, which has a fixed constant $\kappa_i(z) = \kappa_i$. This is the simplest way to choose a step size, and the results from experimenting with this method for different values of $\kappa_i$ should give intuition as to how sensitive the HCMS algorithm is to the choice of step size. Secondly, we attempt to improve on this simple scheme by making a choice of step size based on the Newton method, where a fixed constant $\kappa_i$ is given so that

$$\kappa_i(z) = \kappa_i \Big( \sum_{j \in \mathcal{M}_i} f^2_{j \to i}(p_i(z)) \Big)^{-1},$$

unless this value is negative or above $\kappa_i$, in which case we set $\kappa_i(z) = \kappa_i$. This bounding of $\kappa_i(z)$ deviates from normal Newton method behaviour, however it is necessary to prevent the algorithm from converging to minima, or behaving unpredictably around points of inflection.

The choice of these parameters must be, to some extent, fitted to the problem in question. If the values of $\kappa_i(z)$ are too small, then the algorithm will evolve slowly, and may not reach high quality solutions in the specified number of iterations. If the values of $\kappa_i(z)$ are too large, then the algorithm may be limited in how close it can come to converging on a high quality solution, due to continually overshooting the optimal point. We examine how the performance of the fixed step size gradient and Newton based methods depend on the parameter $\kappa_i$ in Section 5.5.1. We find that there is a wide range of choices which yield good results.

As a rough rule of thumb, we would suggest taking $\kappa_i$ to be at most inversely proportional to an approximate upper bound of:

$$\sum_{j \in \mathcal{M}_i} \left| \frac{d^2 U_j}{d^2 p_i} \right| \tag{5.9}$$

The reason for this is, when using a gradient method to converge to maximise an objective function $f$, if the step size parameter is always chosen to be less than $2/K$, where $K$ is an upper bound on $f''$, then each iteration leads to an improved solution. For example, in the target classification domain (see Section 5.5), we found that 99% of evaluated values of $|d^2 U_j / d^2 p_i|$ were bounded by 2.7. From the number of agents in our experiments Equation 5.9 prescribes an order of magnitude for $\kappa_i$ around 0.1 or 0.01. This is born out by our empirical results, where $\kappa_i$ larger than 0.1 begins to yield poorer results, and $\kappa_i = 0.01$ gives the best results over all.

### 5.4.2 Theoretical Results

We now show some theoretical results that apply to the HCMS algorithm over problems with acyclic factor graphs.

**Proposition 5.1.** *Suppose we apply the HCMS algorithm to an acyclic factor graph. If the step size is decreasing, and is always sufficiently small, then the maximum achievable utility given the set of possible states for each variable strictly increases over time.*

For every iteration, the message passing algorithm acts like the standard max-sum algorithm for the current variable state space discretisations. Thus, once all messages have been sent, by the results in Kschischang et al. (2001) for the standard max-sum algorithm, for each agent $i$, for $z = 1, \ldots, k_i$,

$$\sum_{j \in \mathcal{M}_i} r_{j \to i}(a_i^z) = \max_{\mathbf{p} \setminus p_i, p_i = a_i^z} \sum_{j \in \mathcal{M}_i} U_j(\mathbf{p}_j).$$

For each variable $p_i$ let $a_i^*$ be defined as:

$$a_i^* = \arg\max_{a \in \mathcal{D}_i} \sum_{j \in \mathcal{M}_i} r_{j \to i}(a).$$

By definition, for $\mathbf{a}^* = \{a_1^*, \ldots, a_M^*\}$ it holds that:

$$\mathbf{a}^* = \arg\max_{\mathbf{a} \in \mathcal{D}_\mathbf{p}} \sum_{i=j}^{M} U_j(\mathbf{a}_j).$$

where $\mathcal{D}_{\mathbf{p}}$ is the Cartesian product of the domains of variables $\mathbf{p}$, and $\mathbf{a}_j$ are the elements of $\mathbf{a}$ corresponding to variables $\mathbf{p}_j$. Furthermore, for each variable $p_i$, the value of

$$\sum_{j \in \mathcal{M}_i} f^1_{j \to i}(a)$$

is equal to the partial derivative of the objective function by $p_i$, evaluated at $a \in \mathcal{D}_i$.

Hence, each update step moves each element in the domain $\{a_i^z\}_{z=1}^{k_i}$ of $p_i$ in the direction of the gradient of the objective function evaluated at that point. Provided the step sizes are sufficiently small, then utility at this point will strictly increase (see, for example Avriel (1999) chapters 8 and 9). Thus, after each iteration, there is a combination of states which gives more utility than the previously maximum possible. □

As a corollary to this proposition, we can deduce that the utility of the solution provided by the HCMS algorithm can be made to be arbitrarily close to optimal, if the initial state space discretisations are sufficiently fine. This is because with a sufficiently fine discretisation, then there will be at least one combination of initial possible states which is already sufficiently close to the optimal solution, and the progress of the algorithm can only improve upon this.

### 5.4.3 Communication and Computation Cost

The HCMS algorithm involves a slightly increased communication and computation overhead compared to the discrete max-sum algorithm. Specifically, the differences are as follows:

- Messages passed from function $U_j$ to variable $x_i$ include $f^1_{j \to i}(\cdot)$, and possibly $f^2_{j \to i}(\cdot)$ (Equation 5.8), instead of just $r_{j \to i}(\cdot)$ (Equation 5.3). This results in an increase in communication cost of at most a factor of three.
- Messages passed from variable $x_i$ to function $U_j$ include the updated domain $\{a_i^1, \ldots, a_i^{k_i}\}$, in addition to $q_{i \to j}(\cdot)$ (Equation 5.2). This results in an increase in communication cost of a factor of two.
- In terms of additional computation overhead, for each $z = 1 \ldots k_i$, $f^1_{i \to j}(a_i^z)$, and $f^2_{i \to j}(a_i^z)$ may be calculated using three evaluations of $U_i$ (if there is no fast closed form expression for these derivatives). However, these extra function evaluations do not represent a significant computational cost compared to the optimisation used to calculate the $r_{j \to i}(\cdot)$ functions, in which $U_j$ is evaluated for the entire discrete state space.

So, the increase in communication and computation costs in operating the HCMS algorithm compared to the standard discrete max-sum algorithm is at most a factor of three

and does not depend on the number of agents. Thus, the HCMS algorithm has the same general scalability properties as the discrete max-sum.

However, it is worth noting that the above result applies when comparing the two algorithms operating for the same number of iterations. From Proposition 5.1, we might expect that it is beneficial to operate the HCMS algorithm for more iterations than the discrete max-sum algorithm. In Subsection 5.5.1, we empirically explore how the performance of the HCMS approach is affected by how many iterations are run, and compare this to the behaviour of the discrete max-sum algorithm.

## 5.5 The Target Classification Domain

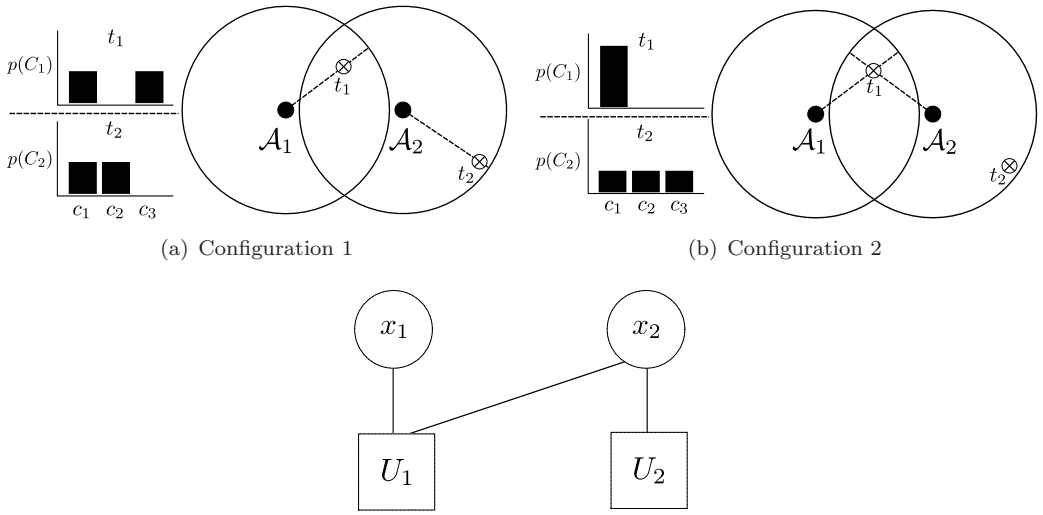
In this section, we empirically evaluate the HCMS algorithm as a means of coordinating agents in the target classification domain. This domain is similar to the one we used in Section 5.3. but, instead of merely detecting events, the agents' goal is now to classify targets. Moreover, we no longer assume that their operation is energy constrained, or that they can monitor their observation area with uniform accuracy. Instead, they are equipped with a sensor whose orientation they control (such as a camera, for example); targets within a small angular distance from the viewing orientation are more likely to be accurately classified than targets that are further away. Furthermore, agents have different capabilities that enable them to classify some classes of targets more accurately than others.

The observation value within this domain is measured in terms of the certainty with which targets are classified. Recall from Section 2.4 that entropy is a common way of formalising certainty, and will be used in this section. Due to the complex interactions between the actions of the agents and this metric, the utility functions are no longer (piecewise) linear, which is a clear advantage in the context of evaluating the HCMS algorithm.

Consider the following example of this problem domain.

**Example 5.4.** *Figures 5.10(a) and 5.10(b) show two scenarios of the target tracking domain. In both scenarios, there are two targets of class $c_1$ and two agents. Agent $\mathcal{A}_1$ is capable of classifying targets of class $c_2$, but is unable to distinguish between classes $c_1$ and $c_3$. Similarly, agent $\mathcal{A}_2$ detects targets of class $c_3$, but can not distinguish between $c_1$ and $c_2$. In Figure 5.10(a), $\mathcal{A}_1$ is directed towards $t_1$, and $\mathcal{A}_2$ towards $t_2$. Given this configuration, the posterior probability distribution over the class of $t_1$ and $t_2$ is shown on the left in Figure 5.10(a). If, however, the agents are configured as in Figure 5.10(b), no information is gained about $t_2$'s class, but $t_1$'s class is correctly determined.*

In more detail, we consider a network of agents $\mathbf{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_M\}$ and a group of targets $T = \{t_1, \ldots t_n\}$ with a random spatial distribution. Targets are assumed to




(a) Configuration 1

(b) Configuration 2

(c) Factor graph for the agent deployment in (a) and (b).

FIGURE 5.10: Two example scenarios and the factor graph to represent them.

be stationary, and can be one of $C = \{c_1, c_2, \dots\}$ classes. Agents are able to take (imprecise) measurements of targets within a fixed sensing range, and are able to rotate their sensor to change its viewing direction. When this sensor is directed at a target, the probability of correctly classifying it is maximised, but when rotated away from a target, the agent acquires less information about the target, and this probability is reduced.

More formally, given a target $t$ whose (unknown) class is modelled as random variable $\mathcal{C}_t$ (with domain $C$), agent $\mathcal{A}_i$ obtains a measurement, modelled as random variable $\mathcal{M}_i$ (also with domain $C$), based on its type and viewing direction. For each agent/target pair, the probability of $\mathcal{A}_i$ classifying a target as $\mathcal{M}_i$, conditioned on its actual class $\mathcal{C}_t$ is given by $p(\mathcal{M}_i\mathcal{C}_t, \theta)$, where $\theta$ is the angle between the agent's viewing direction and target $t$, which ranges between 0 and $\pi$. For $\theta = \pi$ (i.e. the agent looks away from the target), $p(\mathcal{M} \mid \mathcal{C}_t, \theta)$ is a uniform probability distribution over $C$, encoding the fact that no information about the target's class is gained. The following equation has the desired properties:

$$p(\mathcal{M}_i \mid \mathcal{C}_t, \theta) = (1 - f(\theta))\, p^*(\mathcal{M}_i \mid \mathcal{C}_t) + f(\theta)\frac{1}{|C|} \tag{5.10}$$

Here, $p^*(\mathcal{M}_i \mid \mathcal{C}_t)$ is agent $\mathcal{A}_i$'s optimal sensing signature, which applies when $\theta = 0$, and $f(\theta)$ is some function of $\theta$ with $f(0) = 0$ and $f(\pi) = 1$, such that when $\theta = \pi$, $p(\mathcal{M}_s \mid \mathcal{C}_t, \theta)$ is a uniform distribution, as required.

Now, given this, the goal of a team of agents is to minimise the remaining uncertainty in the classification of the targets after the having taken measurements. This is equal to the conditional entropy $H(\mathcal{C}_1, \dots, \mathcal{C}_m \mid \mathcal{M}_1, \dots, \mathcal{M}_n)$ of the target's classes given that the measurements of all agents are known. Since the classes of any two targets $(t, t')$ are assumed to be independent, $H(\mathcal{C}_t, \mathcal{C}_{t'} \mid \mathcal{M}) = H(\mathcal{C}_t \mid \mathcal{M}) + H(\mathcal{C}_{t'} \mid \mathcal{M})$, and the problem is reduced to minimising a sum of conditional entropies of the classification of

individual targets. For an individual target $t$ and a set of agents $\mathbf{A}$ that are in range, the conditional entropy of $\mathcal{C}_t$ given $\mathcal{M}_{\mathbf{A}}$ is given by:

$$\begin{aligned} H(\mathcal{C}_t \mid \mathcal{M}_{\mathbf{A}}) &= \sum_{\mathbf{m}\in\mathbf{M}} H(\mathcal{C}_t \mid \mathcal{M}_{\mathbf{A}} = \mathbf{m}) \\ &= \sum_{\mathbf{m}\in\mathbf{M}, c\in C} p(\mathbf{m}, c) \log \frac{p(\mathbf{m}, c)}{p(m)} \\ &= \sum_{\mathbf{m}\in\mathbf{M}, c\in C} p(\mathbf{m} \mid c)p(c) \log \alpha p(\mathbf{m} \mid c)p(c) \end{aligned} \tag{5.11}$$

where $\mathbf{M} = \times_{\mathcal{A}_i\in\mathbf{A}} M_i$ denotes the set of all possible measurements that agents $\mathbf{A}$ can collectively make, $\alpha$ is a normalising constant, and $p(c)$ is a prior over the target, which is assumed to be a uniform distribution.

Since the viewing angle of an agent is a continuous parameter, taking values from $[0, 2\pi]$, this problem is a distributed optimisation problem with continuous state spaces. Given this, and the fact that the agents' actions interact in non-linear ways, this domain is particularly suitable for benchmarking the HCMS algorithm against existing discrete ones.

In order to instantiate the general problem defined Section 5.6, and to use our HCMS algorithm (as well as the discrete max-sum algorithm), we now show how to build a factor graph for this problem. Firstly, we assign a continuous variable $p_i$ to agent $\mathcal{A}_i$ representing its viewing direction, ranging from 0 to $2\pi$. Secondly, for each *target* $t_j$, we define a function $U_j(\mathbf{p}_j)$ with parameters $p_i \in \mathbf{p}_j$ iff target $t_j$ is in range of agent $\mathcal{A}_i$. Thus, $U_j$ is a continuous function of the agents' viewing directions and is equal to the conditional entropy $H(\mathcal{C}_j \mid \{\mathcal{M}_i \mid p_i \in \mathbf{p}_j\})$ given these viewing directions of agents in range as in Equation 5.11. Thirdly and finally, to obtain a truly decentralised approach, we assign the responsibility of computing the outgoing messages for $U_j$ (Equation 5.3) to one of the agents $\{\mathcal{A}_i \mid p_i \in \mathbf{p}_j\}$ in range, while taking care that the computation load is balanced over these agents.[10] For the simple scenarios in Figures 5.10(a) and 5.10(b), the factor graph is shown in Figure 5.10(c).

### 5.5.1 Experimental Results

We benchmark our algorithm against five algorithms:[11]

[10] Note that, by so doing, we slightly deviate from the problem description in Section 5.1. In this problem description, a utility function $U_i$ is associated with an *agent* $\mathcal{A}_i$, not *target* $t_i$. However, this is merely a naming issue, which can be resolved by summing the utility functions assigned to a single agent and renaming the result. Note also that we took a slightly different approach in Equation 5.7, where we assigned the observation value of an area to each of the agents capable of monitoring it, while avoiding double counting. Both approaches are correct, and illustrate that a coordination problem can be encoded as a factor graph in various ways.

[11] Since the functions in this domain are not piecewise linear, we were unable to benchmark against the CPLF-MS algorithm.

**Discrete Max-Sum** We directly apply the discrete max-sum algorithm to the factor graph constructed above.

**Local Greedy** This algorithm selects the angle that minimises entropy on targets within range, regardless of the angles of its neighbours. This algorithm shows the performance that can be achieved without coordination.

**Distributed Stochastic Algorithm (DSA)** (Fitzpatrick & Meertens 2003) This is an iterative best-response algorithm. Agents are randomly activated and update their angle such that the entropy of targets within range is minimised, while fixing the current angle of their neighbours. DSA is an alternative to discrete max-sum, propagates information more locally, and has been shown to be outperformed by max-sum in general settings (Farinelli, Rogers, Petcu & Jennings 2008).

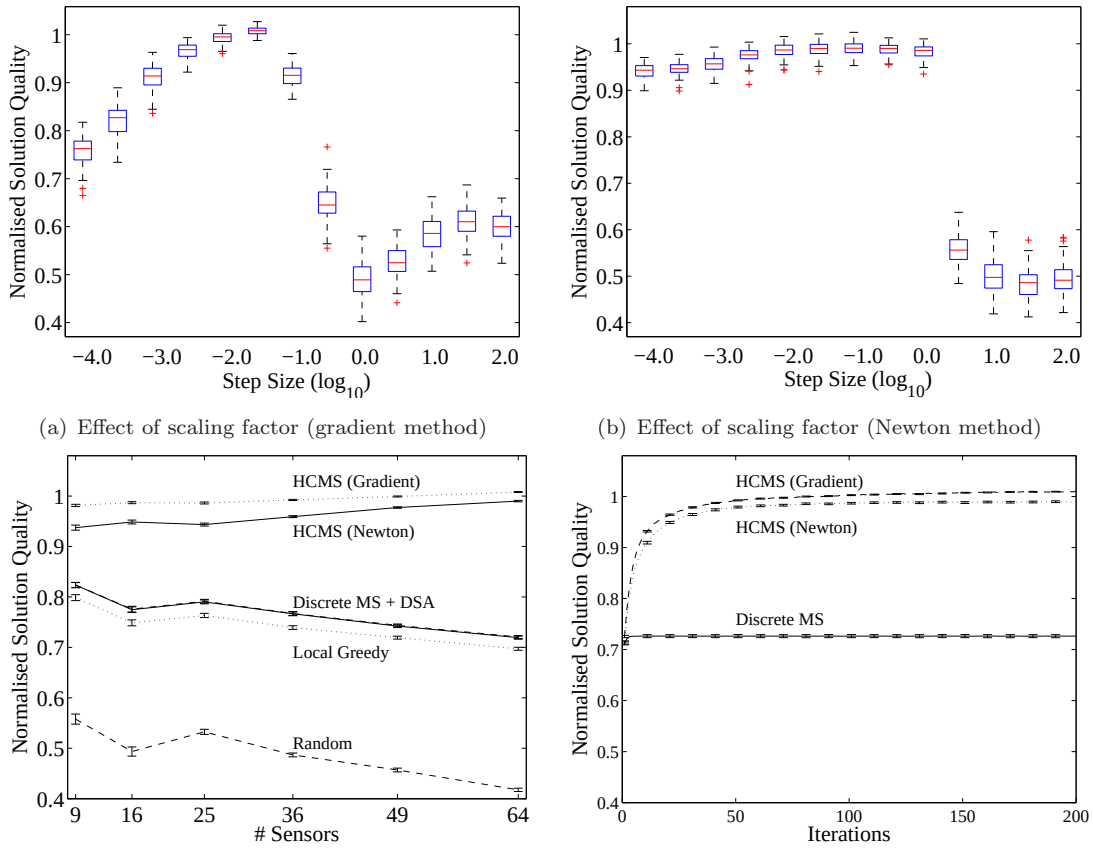
**Random** For each agent, this algorithm selects a viewing angle at random. The random algorithm is included to provide a lower bound on achievable performance.

**Centralised Simulated Annealing** As in Section 5.3.2, we use centralised simulated annealing to normalise solution quality and as an upper bound for achievable performance.

For our experimental evaluation, the agents' variable domain is discretised into 5 angles for the algorithms with a discrete state space (i.e. discrete max-sum, DSA and Local Greedy). The HCMS algorithm starts with the same initial discretisation as the discrete max-sum algorithm. We considered problem instances in which the agents are laid out in a square lattice formation in a unit square environment, consisting of $M = k^2$ agents, with $k \in [3, 8]$, and the range of each agent is chosen as $\frac{1}{k}$ to ensure the agents' ranges are overlapping (but not the extent that the coordination problem becomes so dense that coverage of all targets is trivially ensured). We then randomly generated 100 problem instances (i.e. target locations) for each lattice formation.

First, we tuned the scaling factor $\kappa_i$ for the gradient and Newton method, as discussed in Section 5.4.1. The results are shown in Figures 5.11(a) and 5.11(b), where solution quality is expressed as a fraction of the solution quality provided by simulated annealing. These figures clearly show that the Newton method is much less sensitive to the chosen value of $\kappa_i$ than the gradient method. However, the gradient method, if properly tuned, gives slightly better results.

Second, we took the best gradient ($\kappa_i = 10^{-1.5}$) and Newton ($\kappa_i = 10^{-1}$) variants of our HCMS algorithm and benchmarked them against the discrete algorithms. The results are shown in Figure 5.11(c), and indicate that our hybrid max-sum algorithm outperforms the discrete coordination algorithms (DSA and discrete max-sum) by up to 30%. Moreover, and more importantly, the normalised solution quality shows that our decentralised algorithm performs comparably to the centralised simulated annealing algorithm.




(a) Effect of scaling factor (gradient method)

(b) Effect of scaling factor (Newton method)

(c) Average performance of evaluated algorithms

(d) Speed of convergence of the gradient and Newton variants of HCMS compared to discrete max-sum.

FIGURE 5.11: Empirical results for the HCMS algorithm.

Finally, we evaluated the speed of convergence of the gradient and Newton variants of HCMS on an 8 by 8 lattice formation, as compared to the discrete max-sum algorithm. The results are shown in Figure 5.11(d). This shows that, while discrete max-sum converges more quickly than HCMS, the solution quality of HCMS variants grows much faster over time. Around 20 iterations, both HCMS variants achieve a solution quality that is 30% better than discrete max-sum. However, since the gradient method exchanges the first derivative, and the Newton method both the first and second derivative (see Section 5.4.1), this comes at a cost of a twofold and threefold increase in message size respectively. In terms of computation, this 30% improvement requires twice as many iterations as discrete max-sum (which converges around 10 iterations). Combined with the results in Section 5.4.3, which state that at most 3 evaluations of $U_i$ are required to calculate the additional messages, HCMS requires at most 6 times as much computation as discrete max-sum.

## 5.6 Summary

In this chapter, we presented two novel decentralised algorithm—CPLF-MS and HCMS—for multi-agent coordination problems that are characterised by continuously valued decision variables. Within these settings, the advantage of the two algorithms presented in this chapter over the standard (discrete) max-sum algorithm (and, indeed any discrete optimisation algorithm), is that their use does not involve a artificial trade-off between *scalability* and *quality* that results from selecting a suitable level of discretisation for the variables' domains. For discrete algorithms, the discretisation should be sufficiently coarse to keep the computation tractable (for discrete max-sum, the computation an agent needs to perform is exponential in the number of variable states), while it must be sufficiently fine to enable the algorithm to find high quality solutions. The latter is especially true when the global utility function is very 'peaked', i.e. when the magnitude of its first derivative is large. The algorithms presented in this chapter do not suffer from either drawback.

The first algorithm, CPLF-MS, uses techniques from computational geometry to derive exact algorithmic solutions for performing the two key mathematical operations required by max-sum for continuous piecewise linear functions. We benchmarked CPLF-MS against the standard max-sum algorithm and a centralised simulated annealing algorithm, and found that it outperforms the former by up to 10%, and yields solutions close to the optimal solution computed by the latter. However, we also found that the complexity inherent in using simplexes to represent the utility functions tends to scale unfavourably with the number of neighbouring agents.

The second algorithm, HCMS, avoids these problems. It uses non-linear optimisation techniques to evolve the variable domains used by the standard max-sum algorithm to enable the latter to yield near-optimal solutions. A comparison to the standard max-sum algorithm shows that HCMS improves solution quality by up to 30%, at the cost of a threefold increase in the size of the messages. Moreover, with a sufficient number of iterations, it performs comparably to the centralised simulated annealing algorithm.

In terms of the design requirements stated at the start of this thesis, we can conclude the following:

**Quality:** The experimental results reported in Sections 5.3.2 and 5.5.1 conclusively demonstrate the superiority of CPLF-MS and HCMS over the standard max-sum algorithm in terms of the quality of the achieved situational awareness.

**Adaptiveness:** The adaptiveness of these algorithms for coordinating fixed agents depends primarily on the way they are used; when used for one-off optimisation agents are less capable of adapting to their environment than when the algorithms are run periodically or continuously. In this chapter, we demonstrated how these

algorithms can be used to solve what are essentially snapshots of the wide area surveillance problem. However, when this is done repeatedly, the agents are able to continuously adapt to their environment.

**Robustness:** Both algorithms extend the max-sum algorithm, and as such, inherit its robustness against messages loss and failing agents.

**Autonomy:** Idem as *robustness*.

**Scalability:** The use of simplexes in the CPLF-MS algorithm to represent utility function causes it to scale poorly with the number of neighbours. The HCMS algorithm does not suffer such an unfavourable complexity increase; the use of HCMS involves a constant factor complexity increase compared to the max-sum algorithm, both in terms of computation and communication.

**Modularity:** The action variables of agents whose controls are inherently continuous need no longer be discretised in order to apply the max-sum algorithm. This results in a clear improvement of the modularity of the team of agents, since a larger variety of agents are now able to interoperate.

**Performance Guarantees:** Neither CPLF-MS, nor HCMS provide performance guarantees in the form we have presented them here. However, the bounded max-sum techniques proposed by Farinelli et al. (2009) and discussed in Section 2.6.1 can be directly applied to allow these algorithms to give performance guarantees.

Up until this point, we have studied the challenges inherent in coordinating fixed agents in two phases of their lifetime: deployment and operation. In the next two chapters, we shift our focus to mobile agents, and the challenges involved in coordinating their movements in order to maximise observation value. As we will see, the coordination problem this poses is more dynamic than the ones we have seen so far. This is due to the fact that every time the agents reposition themselves, a new coordination problem arises, which has different utility functions and action spaces than the one before. Moreover, this problem is more demanding, because the action spaces of individual agents are more complex. Nevertheless, we show that the max-sum algorithm, which formed the basis for the two algorithms we developed in this chapter, can also be adopted to solve this problem.

# Chapter 6

# Decentralised Receding Horizon Control of Mobile Agents

In this chapter,[1] we turn to the challenge of coordinating mobile agents. Their mobility allows these agents to observe more of their environment over time than the same number of fixed agents are able to. Given this, mobile agents can provide equivalent situational awareness in small numbers, to that of their fixed counterparts in larger numbers. However, the smaller scale of the teams in which they operate does not necessarily simplify the challenge of coordinating their actions. Typically, the motion constraints imposed by the layout of the environment (which is encoded by a graph $G$ as per the problem formulation in Chapter 3) increase the complexity of their action space as compared to the fixed agents we have considered in previous chapters. As a result, care must be taken to ensure the *scalability* of a coordination algorithm, while preserving the *quality* of situational awareness it brings about.

Against this background, in this chapter we develop an efficient decentralised coordination algorithm that plans the agents' movements over a *receding horizon*. This means that agents periodically coordinate to maximise the observation value received as a team

| Chapter | Type of Agent | Planning Horizon | Lifecycle Phase |
|---|---|---|---|
| 4 | Fixed | 1 | Deployment |
| 5 | | | Operation |
| 6 | Mobile | $m$ | |
| 7 | | $\infty$ | |

TABLE 6.1: The contributions of Chapter 6 in the context of the roadmap of this thesis.

[1] This chapter is based on Stranders et al. (2009*b*) and Stranders, Delle Fave, Rogers & Jennings (2010).

for a fixed number of time steps $l$ in the future. Using this algorithm, agents coordinate their plans (i.e. finitely long paths in $G$), which they implement for $m \leq l$ time steps. After this, they coordinate again to plan their motion for the next $l$ time steps. As a result, the planning intervals of two subsequent executions of the algorithm overlap. This enables the agents to plan ahead, while still being able to revise their plans based on observing *a priori* unknown events, which results in an adaptive and—as we will show—effective approach.

To implement receding horizon control in a decentralised fashion, we again opt for the max-sum algorithm. The benefits of this algorithm have already been extensively discussed in Section 2.6.1, but are worth reiterating. The use of the max-sum algorithm enables agents to coordinate their information gathering tasks in a fully decentralised and scalable fashion, while being robust against message loss and failure of individual agents.

However, the potentially very large action spaces of the individual agents makes the straightforward application of the max-sum algorithm to this problem infeasible. Specifically, these action spaces consist of paths from the agents' current location, and since the set of all possible paths grows exponentially with the length $l$ of the planning horizon, considering all of these allows us to solve only the smallest of problem instances (in terms of $l$ as well as the number of agents). Therefore, we make three augmentations to make our algorithm more scalable, all of which aim to reduce the action space that needs to be searched in the main bottleneck of the max-sum algorithm (i.e. the computation of messages from function to variable in Equation 2.13). By so doing, we significantly reduce the computational cost that individual agents incur as a function of the number of neighbours, thus improving the algorithm's scalability.

In more detail, the contributions made in this chapter are as follows:[2]

- We develop an accurate and robust decentralised receding horizon coordination algorithm for mobile agents based on the max-sum algorithm. To improve the algorithm's scalability compared to a straightforward application of max-sum, we do the following:
  - We exploit the property of *locality* of the observation value function (see Chapter 3) to reduce the number of dependencies between agents, which results in an exponential reduction of the joint action space agents need to search.
  - We develop two heuristics for defining an individual agent's action space to reduce its action space. Both select a small number of paths from the exponentially large set of all possible paths from the agents' current location.

[2]Table 6.1 shows the context of these contributions in terms of the roadmap of this thesis.

  - We develop two pruning techniques for speeding up the max-sum algorithm. The first aims to reduce the size of the action space of individual agents by removing dominated actions. The second uses branch and bound to reduce the joint action space that needs to be searched. These techniques are general in the context of max-sum, and their application is thus not limited to the multi-agent information gathering problem we address in this thesis.

- We demonstrate the algorithm's domain independence (within the context of the problem formulated in Chapter 3) by evaluating it in three different information gathering domains, and show that:

  - it increases the prediction accuracy by up to 50% in the environmental monitoring domain;
  - it decreases the capture time of an evader by at least 30% in the pursuit-evasion domain;
  - it decreases the damage from intrusion by at least 30% in the patrolling domain,

  compared to an un-negotiated greedy algorithm and the state of the art in agent-based information gathering (Vidal et al. 2001, Hespanha et al. 1999, Sak et al. 2008).

- By applying our algorithm to environmental monitoring, pursuit-evasion and patrolling, we obtain the first online decentralised coordination algorithms for these domains.

The remainder of this chapter is organised as follows. In Section 6.1, we extend the problem formulation in Chapter 3 to the problem of receding horizon control. In Section 6.2 we develop the coordination algorithm for solving this problem. In Section 6.3 we present an extensive empirical evaluation of this problem on three information gathering domains. Finally in Section 6.4 we summarise the contributions of this chapter and analyse them in terms of the design requirements from Chapter 1.

## 6.1 Problem Formulation

The problem that we address in this chapter extends the mobile agent problem formulated in Chapter 3 for receding horizon control. To do this, we limit the number of time steps for which the information value is maximised to a finite number $l$ in Equation 3.2. Thus, at time $t$, agents should plan their motion for the time interval $[t, t + l]$ in order

to maximise the discounted incremental value of the observations they make during this interval:[3]

$$\sum_{t'=t}^{t+l} \gamma^t \rho_{O^{t'}_{\mathbf{A}}}(\mathbf{O}^{t'-1}_{\mathbf{A}}) \tag{6.1}$$

For every agent, this yields a path of length $l$ from their current position. This computation is performed every $m \leq l$ time steps, and thus, recomputation can occur before the agents have entirely traversed these paths.

This approach does not yield optimal solutions to the NP-hard problem of maximising discounted observation value maximisation in Equation 3.2. However, even the receding horizon control problem is NP-hard[4], and thus, we need to resort to (further) approximation. Thus, instead of computing optimal solutions to the receding horizon control problem, which involves computing the value of an exponential number of paths of length $l$, the algorithm we develop in this chapter trades off solution quality for a significantly reduced computational overhead.

## 6.2 The Coordination Algorithm

As in the previous chapter, we use the max-sum algorithm as a baseline solution for maximising observation value in a decentralised fashion. In order to use the max-sum algorithm to achieve receding horizon control, and thus maximise Equation 6.1, we proceed in three steps:

1. We decompose Equation 6.1 into a sum of utility functions, each of which encodes how much observation value a single agent contributes to the total observation value collected as a team.

2. We assign a single action variable to each agent representing the path it will follow for the next $l$ time steps. Thus, the domain of these variables consist of paths of length $l$ from the corresponding agent's current location. We develop two heuristics for limiting the size of these—otherwise exponentially large—domains.

3. We apply the max-sum algorithm to a factor graph constructed from the utility functions and and variables defined in the previous steps. We develop two pruning techniques to further reduce computational overhead, especially in settings where utility functions are costly to evaluate.

[3] In this chapter, we assume $\gamma = 1$ (i.e. no discounting of observation value). This is done in the interest of simplifying notation, and does not restrict the application of the algorithm to problems where this assumption does not hold.

[4] The NP-hard problem of maximising a submodular function can be reduced to this problem, which is essentially maximising a submodular function with additional non-trivial constraints.

In the three subsections that follow, we will further detail these steps.

### 6.2.1 Factorising the Value of Observation

The first step towards making the receding horizon control problem amenable to optimisation by the max-sum algorithm is factorising Equation 6.1 into a set of agent-dependent utility functions. To do this, we need to factorise the incremental value of the observations made by all agents in the next $t$ time steps. We will denote this set as $O_{\mathbf{A}}$, which is equal to the union of observations made by each individual agent at each of the next $t$ time steps: $O_{\mathbf{A}} = \bigcup_{t'=t}^{t+l} \bigcup_{i=1}^{M} O_i^t$.

A naïve factorisation—that, as it turns out, is incorrect—is to consider the value of only those observations that an agent will make, since observation across agents are generally not independent. Thus, due to the submodularity property of the value function $f$, this method overestimates the true value that is obtained:

$$f(O_1 \cup \ldots \cup O_M \cup \mathbf{O}_{\mathbf{A}}^{t-1}) - f(\mathbf{O}_{\mathbf{A}}^{t-1}) \leq \sum_{i=1}^{M} f(O_i \cup \mathbf{O}_{\mathbf{A}}^{t-1}) - f(\mathbf{O}_{\mathbf{A}}^{t-1}) \tag{6.2}$$

with equality if and only if all pairs of observation sets $O_i$ and $O_j$ are independent. (Recall from Chapter 3 that this is the case when the observations contained in these sets are further than $\delta$ apart through the property of *locality*. We come back to this shortly.) However, this is unlikely to be the case in practise. As a result, if each agent attempts to maximise the observation value it receives in an uncoordinated fashion, it is unlikely they will maximise the observation value received as a team.

Instead, we decompose the team utility in Equation 6.1 into a set of factors, one for every agent, such that these factors sum to the true observation value received by the team. In order to do this, note that:

$$\begin{aligned} f(O_1 \cup \ldots \cup O_M \cup \mathbf{O}_{\mathbf{A}}^{t-1}) - f(\mathbf{O}_{\mathbf{A}}^{t-1}) &= \sum_{i=1}^{M} f\left(\bigcup_{j=1}^{i} O_j \cup \mathbf{O}_{\mathbf{A}}^{t-1}\right) - f\left(\bigcup_{j=1}^{i-1} O_j \cup \mathbf{O}_{\mathbf{A}}^{t-1}\right) \\ &= \sum_{i=1}^{M} \rho_{O_i}\left(\bigcup_{j=1}^{i-1} O_j \cup \mathbf{O}_{\mathbf{A}}^{t-1}\right) \end{aligned} \tag{6.3}$$

Equation 6.3 states that the team utility is a sum of the *incremental values* (see Definition 2.3) obtained by adding observations $O_i$ to the observations made by agents $\mathcal{A}_j$, $j < i$. We will call an individual factor of this sum the *contribution* of an agent, or its *utility*.

**Definition 6.1** (Agent Contribution/Utility)**.** The contribution agent $\mathcal{A}_i$ makes to the team utility by observing $O_i$, conditioned on the fact that agents $j < i$ make observations

$O_1, \ldots, O_{i-1}$ is:

$$U_i(O_1, \ldots, O_i) = \rho_{O_i}\left(\bigcup_{j=1}^{i-1} O_j \cup \mathbf{O}_\mathbf{A}^t\right)$$

Thus, in order to calculate its contribution, an agent need only be aware of the samples collected by agents with a lower index. This still poses a problem, because these agents might not be in communication range, and creates a large number of dependencies between the agents (to be precise, this implies $\frac{1}{2}\binom{M}{2}$ dependencies, among $M$ agents). Fortunately, further simplifications to an agent's contribution function can be made if we take into account that some observations are independent through the property of *locality* defined in Chapter 3. For example, if the observations $O_k$ of some agent $\mathcal{A}_k$, $k < i$, are independent from observations $O_i$, then:

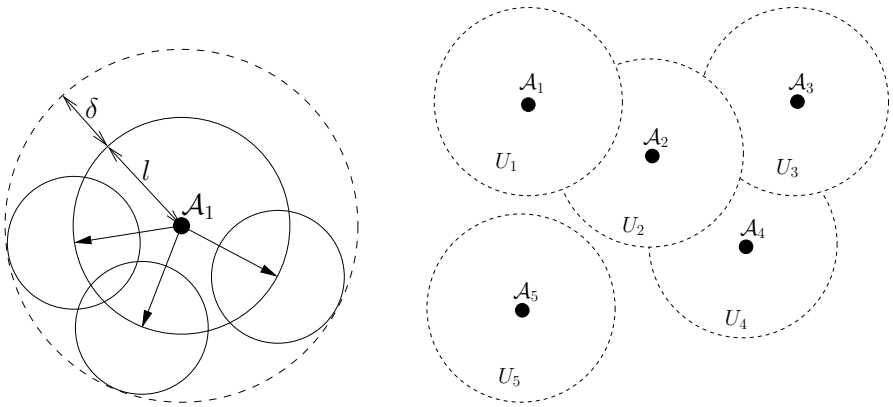
$$\begin{aligned}
\rho_{O_i}\left(\bigcup_{j=1}^{i-1} O_j \cup \mathbf{O}_\mathbf{A}^{t-1}\right) &= f(O_1 \cup \ldots \cup O_i \cup \mathbf{O}_\mathbf{A}^{t-1}) - f(O_1 \cup \ldots \cup O_{i-1} \cup \mathbf{O}_\mathbf{A}^{t-1}) \\
&= f(O_1 \cup \ldots \cup O_{k-1} \cup O_{k+1} \cup \ldots \cup O_i \cup \mathbf{O}_\mathbf{A}^{t-1}) \\
&\quad - f(O_1 \cup \ldots \cup O_{k-1} \cup O_{k+1} \cup \ldots \cup O_{i-1} \cup \mathbf{O}_\mathbf{A}^{t-1}) \\
&= \rho_{O_i}\left(\bigcup_{j=1, j\neq k}^{i-1} O_j \cup \mathbf{O}_\mathbf{A}^{t-1}\right) \qquad (6.4)
\end{aligned}$$

Thus, if observations made further apart than $\delta$ can be considered independent, and $l$ is the maximum distance an agent can travel within the planning horizon, observations made outside a circle of radius $d+l$ centred at the agent's current location are necessarily independent from any observation made by the agent (which are taken within the circle with radius $l$). We refer to this circle as the agent's *influence circle*, of which an example is given in Figure 6.1(a)). Thus, no dependencies exists between agents outside an agent's influence circle.

To clarify this issue further, the following example illustrates the ideas developed in this section.

**Example 6.1.** *Depicted in Figure 6.1(b) are five agents with their respective influence circles. Table 6.2 shows the contribution functions $U_i$ of each agent to the team utility. Using Equation 6.4, the expression for their contribution functions can be further simplified, based on whether the agents' influence circles overlap.*

So far, we have shown how the team utility can be decomposed into a sum of agent utility functions. For agent $\mathcal{A}_i$, this contribution is calculated by determining the *incremental value* of adding observations $O_i$ to the set of observations collected by agents with a




(a) Influence circle of a single agent (explanation in text).

(b) An example scenario with 5 agents.

FIGURE 6.1: The *influence circles* of mobile agents.

| Agent | $U_i$ | Simplified $U_i$ |
|---|---|---|
| $\mathcal{A}_1$ | $\rho_{O_1}\left(\mathbf{O}_\mathbf{A}^{t-1}\right)$ | Simplification not possible |
| $\mathcal{A}_2$ | $\rho_{O_2}\left(O_1 \cup \mathbf{O}_\mathbf{A}^{t-1}\right)$ | Simplification not possible |
| $\mathcal{A}_3$ | $\rho_{O_3}\left(O_1 \cup O_2 \cup \mathbf{O}_\mathbf{A}^{t-1}\right)$ | $\rho_{O_3}\left(O_2 \cup \mathbf{O}_\mathbf{A}^{t-1}\right)$ |
| $\mathcal{A}_4$ | $\rho_{O_4}\left(O_1 \cup O_2 \cup O_3 \cup \mathbf{O}_\mathbf{A}^{t-1}\right)$ | $\rho_{O_4}\left(O_2 \cup O_3 \cup \mathbf{O}_\mathbf{A}^{t-1}\right)$ |
| $\mathcal{A}_5$ | $\rho_{O_5}\left(O_1 \cup O_2 \cup O_3 \cup O_4 \cup \mathbf{O}_\mathbf{A}^{t-1}\right)$ | $\rho_{O_5}\left(\mathbf{O}_\mathbf{A}^{t-1}\right)$ |

TABLE 6.2: Contribution functions for the scenario in Figure 6.1(b).

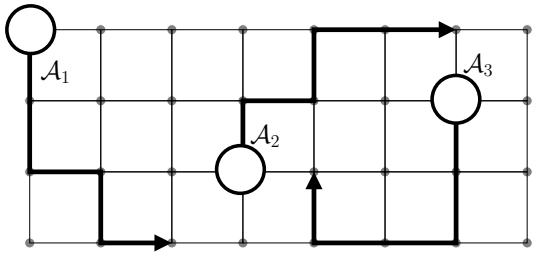



FIGURE 6.2: Three paths of length $l = 5$ for three agents on a lattice graph.

lower index, and the observations that were collected previously. In the next section, we address how the movement constraints imposed by the environment affect which observations are made by the agents.

### 6.2.2 The Action Model

Recall from Chapter 3 that the movement of an agent is subject to constraints imposed by layout graph $G$. Thus, the observations an agent can make in the next $l$ time steps have to lie along a *path* in $G$ of length $l$ from the agent's current location.

If we consider all possible paths, for agent $\mathcal{A}_i$, the domain $\mathcal{D}_i$ of its action variable $p_i$ contains all paths to locations reachable within $l$ time steps (an example for three agents

is shown in Figure 6.2). On average, this domain contains $O(d^l)$ paths, where $d$ is the average degree of graph $G$. Thus, this set grows exponentially with the length of the planning horizon $l$. It is even worse if we consider the joint action space of a team of $M$ agents, which also grows exponentially with the number of agents: $O((d^l)^M)$. For example, in the very simple environment of Figure 6.2, there are 712 possible paths of length 5 for agent $\mathcal{A}_2$. Clearly, to achieve scalable solutions it is necessary to reduce the size of this space.

We achieve this using three different techniques. The first is the max-sum algorithm itself, which restricts the search space of a single agent to the joint action space of its neighbours, instead of the joint action space of all agents.[5] The second are the pruning techniques we develop in Section 6.2.4.1. The third are two heuristic that limit the number of paths an individual agent considers.

Both heuristics are based on the fact that many paths are unlikely to result in optimal solutions, (such as paths that return it to its original position in a short period of time), and on the fact that similar paths lead to almost identical observation value (such as two paths leading to the same location via a small but different detour).

- The first heuristic defines the action space of an agent as paths of length $l$ in in eight directions, corresponding to the major directions on the compass rose. Figure 6.3(a) shows an example. The big vertices are the eight locations that can be reachable in $l = 20$ steps in eight different directions from the agent's current location (the white circle). The thick lines (dashed plus solid) are the shortest paths leading to these locations. The paths are shortened to the length of the shortest path (in this case, this length is 6), and duplicates are removed, resulting in 5 distinct paths (solid).

- The second heuristic uses a graph clustering algorithm and information about the current observation value associated with each vertex, and proceeds in three steps (see Figure 6.3(b) for an example):

    1. It clusters the $l$-neighbourhood[6] $N_G(v)$ of the agent's current location $v \in V$ into a set $\{C_1, \ldots, C_c\}$ of $c$ clusters. The agent's possible destinations are sought within these clusters, and should therefore contain promising locations. Here, we use the k-means algorithm to find well-separated clusters. In the example of Figure 6.3(b) the agent's 20-neighbourhood is clustered into $c = 4$ clusters (coloured blue, black, yellow and red).

[5] Thus, this reduces the joint state space from $O((d^l)^M)$ to $O((d^l)^N)$, where $N \ll M$ is the number of neighbours.

[6] The $l$ neighbourhood of a vertex $v$ is the subgraph of $G$ induced by the vertices reachable in $l$ steps from $v$.

2. Within each cluster $C_i$, it identifies the vertex $v \in C_i$ for which $f$ currently reaches a maximum, i.e. the observation made at $v$ is currently the most valuable within that cluster. In Figure 6.3(b), these are the large vertices.
3. It defines the domain of $p_i$ as the set of shortest paths leading to each of these $c$ vertices. The paths are shortened to ensure they are of the same length, and duplicates are removed. In Figure 6.3(b), the resulting three paths of length 7 are indicated by arrows.

Note that the resulting paths are not always of length $l$. In this case, recomputation is needed earlier than $l$ time steps (if $l < m$). In Section 6.3 we empirically determine the effectiveness of these two heuristics.

At this point, we have defined the action space of the agents, as well as their utility functions. What remains is the application of the max-sum algorithm to find the joint action that maximises observation value received over the planning horizon.

### 6.2.3 Maximising the Value of Observation

The central problem is now to find observation sets $O_1^*, \ldots, O_M^*$ that the maximise observation value received by the agents as a team:

$$\{O_1^*, \ldots, O_M^*\} = \arg\max_{O_1,\ldots,O_M} \left[ f(O_1, \ldots, O_M, \mathbf{O}_\mathbf{A}^{t-1}) - f(\mathbf{O}_\mathbf{A}^{t-1}) \right] \tag{6.5}$$

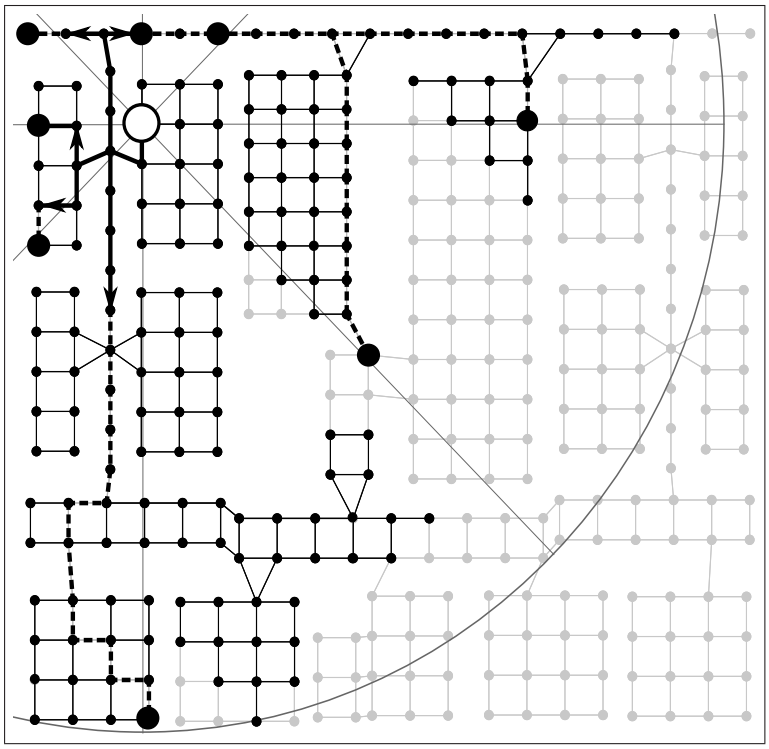
Using Equation 6.3 and Definition 6.1, we can transform this into a sum of agent utilities:

$$\{O_1^*, \ldots, O_M^*\} = \arg\max_{O_1,\ldots,O_M} \sum_{i=1}^{M} U_i(O_1, \ldots, O_i) \tag{6.6}$$

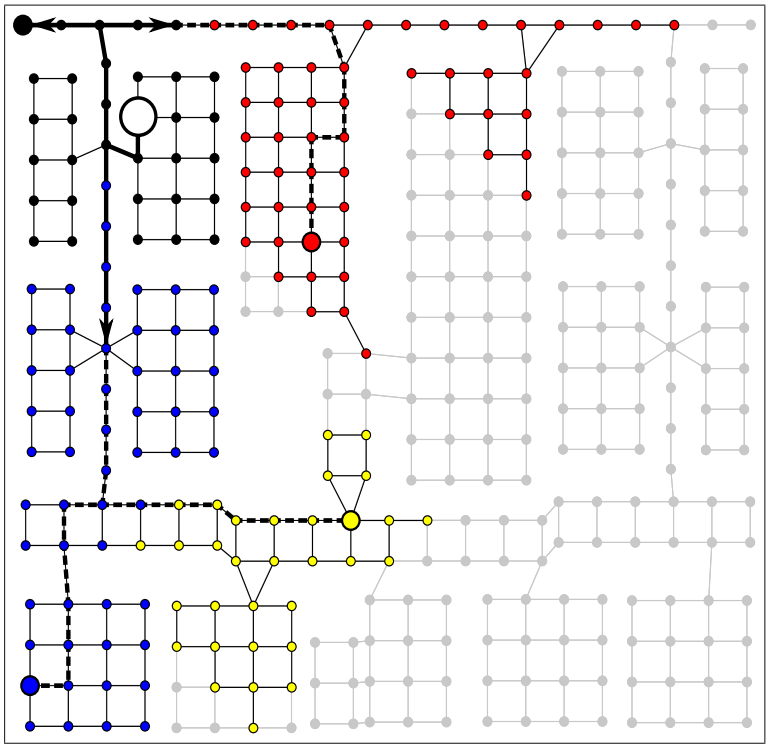
By applying the movement constraints imposed by $G$ encoded in the domains of variables $p_i$ defined in Section 6.2.2, and realising that there exists a correspondence between a path and a set of observations, Equation 6.6 can be transformed into the familiar form of a welfare optimisation problem (Equation 2.9):

$$\mathbf{a}^* = [a_1^*, \ldots, a_M^*] = \arg\max_{p_1,\ldots,p_M} \sum_{i=1}^{M} U_i(p_1, \ldots, p_i) \tag{6.7}$$

As discussed before, we can simplify this equation by exploiting the fact that some observations are independent by applying Equation 6.4. As a result, the number of parameters of the utility functions $U_i$ can be reduced; that is, some of the parameters $p_1, \ldots, p_i$ can be discarded, because the observations of the corresponding agents cannot



(a) The action space identified by heuristic 1.



(b) The action space identified by heuristic 2.

FIGURE 6.3: The action spaces that result from the two action selection heuristics.

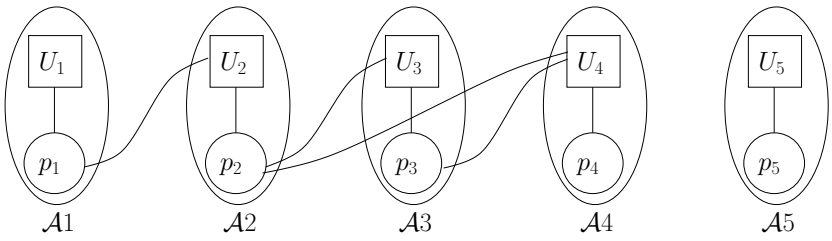



FIGURE 6.4: Factor graph encoding the coordination problem from Example 6.1.

influence agent $\mathcal{A}_i$'s contribution to the team utility. In what follows, we will therefore use the standard max-sum notation $\mathbf{p}_i$ to indicate the vector of parameters to function $U_i$. This vector $\mathbf{p}_i$ contains only those decision variables on which sensor $i$'s utility depends. Since an agent's contribution necessarily depends on the observations it collects itself, $p_i \in \mathbf{p}_i$ will always hold.

### 6.2.4 Applying the Max-Sum Algorithm

We have now transformed the problem so it is amenable to optimisation by the max-sum algorithm; Equation 6.7 is easily translated into a factor graph, on which the max-sum algorithm can be directly applied. Recall from Section 2.6.1 that, in this factor graph, the utility functions $U_j(\mathbf{p}_j)$ are represented by function nodes, and the action variables $p_i$ by variable nodes; an edge exists between $U_j(\mathbf{p}_j)$ and $p_i$ if and only if $p_i \in \mathbf{p_j}$.

The following example shows how Equation 6.7 is encoded as a factor graph for the coordination problem in Example 6.1.

**Example 6.2.** *Figure 6.4 shows an example factor graph that encodes Equation 6.7 for the coordination problem in Example 6.1. In this example, the utility of agent* $\mathcal{A}_1$ *depends solely on its own action, so* $\mathbf{p}_1 = \{p_1\}$*; the the utility of agent* $\mathcal{A}_2$ *depends on its own action, and that of agent* $\mathcal{A}_1$*, so* $\mathbf{p}_2 = \{p_1, p_2\}$*. Similarly,* $\mathbf{p}_3 = \{p_2, p_3\}$*,* $\mathbf{p}_4 = \{p_2, p_3, p_4\}$*, and* $\mathbf{p}_5 = \{p_5\}$*.*

#### 6.2.4.1 Speeding Up Message Computation

Now, while it is possible to apply the max-sum algorithm in its unaltered form, the computation of the messages from function to variable (Equation 2.13):

$$r_{j\to i}(p_i) = \arg\max_{\mathbf{p}_j \setminus p_i} \left[ U_j(\mathbf{p}_j) + \sum_{k \in \mathcal{N}_j \setminus i} q_{k\to j}(p_k) \right] \tag{6.8}$$

can form a major bottleneck that can hamper the *scalability* of the algorithm. The standard way of computing these messages for a given variable $p_i$ is to enumerate all joint

**Algorithm 6** Algorithm for computing pruning message from function $U_j$ to variable $p_i$

1: Compute an upper bound $\overline{U_j}(p_i) \geq \min_{\mathbf{p_j} \setminus p_i} U_j(\mathbf{p}_j)$
2: Compute a lower bound $\underline{U_j}(p_i) \leq \max_{\mathbf{p_j} \setminus p_i} U_j(\mathbf{p}_j)$
3: Send $\langle \overline{U_j}(p_i), \underline{U_j}(p_i) \rangle$ to $p_i$

**Algorithm 7** Algorithm for computing pruning messages from variable $p_i$ to all functions $U_j$ adjacent to $p_i$ in the factor graph.

1: **if** a new message has been received from all $U_j$: $j \in \mathcal{M}_i$ **then**
2: $\quad \bot(p_i) = \sum_{j \in \mathcal{M}_i} \underline{U_j}(p_i)$
3: $\quad \top(p_i) = \sum_{j \in \mathcal{M}_i} \overline{U_j}(p_i)$
4: $\quad$ **while** $\exists a \in \mathcal{D}_i : \top(i) < \max_{p_i} \bot(p_i)$ **do**
5: $\quad\quad \mathcal{D}_i \leftarrow \mathcal{D}_i \setminus \{a\}$
6: $\quad$ **end while**
7: $\quad$ send updated domain $\mathcal{D}_i$ to each $U_j : j \in \mathcal{M}_i$
8: **end if**

paths (i.e. the Cartesian product of the domains of the variables in $\mathbf{p}_j$), and find the one that maximises $U_j$. Since the size of this joint action space grows exponentially with the number of neighbours, the amount of computation can become prohibitive. The reason this has not posed a problem in the previous chapter is that the domains were kept small, and the utility functions were fairly cheap to evaluate. However, evaluating the utility functions in Definition 6.1 can be very costly, since the value of many observations has to be calculated simultaneously.[7] Therefore, we introduce two generic pruning algorithms to reduce the size of the joint action space that needs to be considered. These algorithms are generic in the context of the max-sum algorithm, and can therefore be applied to problems other than multi-agent information gathering.

**The Action Pruning Algorithm** The first algorithm attempts to reduce the number of actions each agent needs to consider *before* running the max-sum algorithm. This algorithm prunes the dominated actions that can never maximise Equation 6.7, regardless of the actions of other agents. More formally, an action $a' \in \mathcal{D}_i$ is dominated if there exists an action $a^* \in \mathcal{D}_i$ such that:

$$\forall \mathbf{a}_{-i} : \sum_{j \in \mathcal{M}_i} U_j(a', \mathbf{a}_{-i}) \leq \sum_{j \in \mathcal{M}_i} U_j(a^*, \mathbf{a}_{-i}) \tag{6.9}$$

where $\mathbf{a}_{-i}$ is an element of the domain of variables $\mathbf{p}_j \setminus p_i$.

[7] Particularly, determining the value of observations using the entropy metric in Section 6.3.1, involves inverting a potentially very large matrix $K(\mathbf{X}, \mathbf{X})$ (see Equation 2.2).

The central idea behind this algorithm is that variable and function nodes in the factor graph collaborate to find lower $\perp(\cdot)$ and upper $\top(\cdot)$ bounds on the value (lines 2–3 of Algorithm 7) of each action $a$, i.e.:

$$\perp(p_i = a) = \min_{\mathbf{a}_{-i} \in \mathcal{D}_{-i}} \sum_{j \in \mathcal{M}_i} U_j(a, \mathbf{a}_{-i})$$

$$\top(p_i = a) = \max_{\mathbf{a}_{-i} \in \mathcal{D}_{-i}} \sum_{j \in \mathcal{M}_i} U_j(a, \mathbf{a}_{-i})$$

Once a variable $p_i$ detects that the upper bound of an action $a' \in \mathcal{D}_i$ is lower than the lower bound of another another action $a \in \mathcal{D}_i$, it removes $a'$ from its domain. Just as with the max-sum algorithm itself, this algorithm is implemented by message passing, and operates directly on the variable and function nodes of the factor graph, making it fully decentralised:

- **From function to variable**: Function $U_j$ sends a message to $p_i$, containing an upper and lower bound on the values of $U_j$ with respect to $p_i = a_i$, for all $a_i \in \mathcal{D}_i$. (see Algorithm 6).
- **From variable to function**: Variable $p_i$ sums the minimum and maximum values from each of its adjacent functions, and prunes dominated actions. It then informs neighbouring functions of its updated domain (see Algorithm 7).

Using this distributed algorithm, functions continually refine the bounds on the utility for a given state of a variable, which potentially causes more actions to be pruned. Therefore, it is possible that action pruning starts with a single action at a single agent, and subsequently propagates through the entire factor graph. This algorithm terminates once the messages exchanged between the functions and variables converge. That is, when all messages along all edges in the factor graph equal the previously received messages. Also note that termination is guaranteed because of the simple fact that every variable has a finite number of states: during each iteration either at least one variable state is pruned or the algorithm has converged. To see why this is true, note that for the bounds on $U_j$ for a certain action to change, at least one variable state needs to get pruned. Otherwise, the messages sent from variables to functions will be identical, and all variables receive the same message twice, which results in the termination of the algorithm.

Given the highly non-linear relations on which the agents' utility functions are often based[8] it is very difficult to calculate tight bounds without exhaustively searching the domain of $\mathbf{p}_j$ for utility function $U_j$. Needless to say, this would defeat the purpose of

[8] Again, entropy is a good example of this, which is based on the variance of a GP expressed in Equation 2.2.

**Algorithm 8** Greedy algorithm for approximating lower bound $\underline{U_j}(p_i)$

1: $\mathbf{p}'_j = \mathbf{p}_j$
2: $U'_j = U_j$
3: **while** $\mathbf{p}'_j \setminus \{p_i\} \neq \varnothing$ **do**
4: Let $p$ be some variable in $\mathbf{p}'_j \setminus \{p_i\}$.
5: $U'_j(\mathbf{p}'_j \setminus \{p\}) \leftarrow \min_p U'_j(\mathbf{p}'_j)$
6: $\mathbf{p}'_j \leftarrow \mathbf{p}'_j \setminus \{p\}$
7: **end while**
8: **return** $U'_j(\mathbf{p}'_j)$ $\{= U'_j(p_i) = \underline{U_j}(p_i)\}$

using this pruning technique. In these cases, we can resort to approximation of these bounds using a greedy algorithm. This algorithm approximates the lower bound $\underline{U_j}(a_n)$ on an action $a_n \in \mathcal{D}_i$ ($p_i \in \mathbf{p}_j$) by iterating through each neighbouring agent, and computing the action that reduces the utility of agent $j$'s action the *most*. This idea is formalised in Algorithm 8; in lines 3–6 the algorithm iterates through the variables, and in line 4 the current variable is "minimised out". This process continues until only $p_i$ is left. The resulting function $U'_j(\mathbf{p}'_j)$ equals the desired bound $\underline{U_j}(p_i)$, which encodes the lower bounds of all actions in the domain of $p_i$. In a similar vein, the upper bounds on a single action is obtained, by selecting those actions of other agents that reduce the utility the *least*. Thus, by substituting max for min in line 4 of Algorithm 8, we obtain an algorithm for approximating $\overline{U_j}(p_i)$

**The Joint Action Pruning Algorithm** Whereas the first algorithm runs as a pre-processing phase to max-sum, the second one is geared towards speeding up the computation of the messages from function to variable (Equation 6.8), *while* max-sum is running. As described earlier, the standard way of computing this message to a single variable $p_i$ is to determine the maximum utility for each of $\mathcal{A}_i$'s actions by exhaustively enumerating the joint domain of its neighbours' variables $\mathbf{p}_j \setminus \{p_i\}$ (i.e. the Cartesian product of the domains of these variables), and evaluating the expression between brackets in Equation 6.8, which we denote by:

$$\tilde{r}_{j\to i}(\mathbf{p}_j) = U_j(\mathbf{p}_j) + \sum_{k \in \mathcal{N}_j \setminus i} q_{k\to j}(p_k) \tag{6.10}$$

However, instead of just considering joint actions, we now allow some actions to be undetermined, and thus, consider *partial* joint actions, denoted by $\hat{\mathbf{a}}$. By doing so, we can create a search tree on which we can employ branch and bound to significantly reduce the size of the domain that needs to be searched. In more detail, to compute $r_{j\to i}(a_i)$ (a single element of the message from $U_j$ to variable $p_i$) for a single action $a_i \in \mathcal{D}_i$, we create a search tree $\mathcal{T}(a_i)$ as follows:

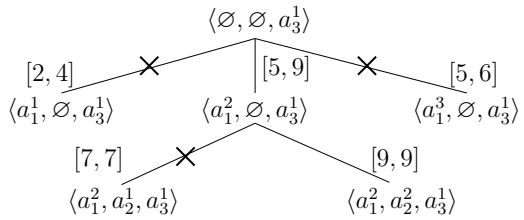



FIGURE 6.5: Search-tree for computing $r_{j\to 3}(a_3^1)$ showing lower and upper bounds on the maximum value in the subtree.

- The root of $\mathcal{T}(a_i)$ is a partial joint action $\hat{\mathbf{a}}_r = \langle \varnothing, \ldots, \varnothing, a_i, \varnothing, \ldots, \varnothing \rangle$, which indicates that $a_i$ is assigned to $p_i$, and the remaining variables are unassigned (denoted by $\varnothing$).
- The children of a vertex $\left\langle a_1^{(1)}, \ldots, a_k^{(k)}, \varnothing, \ldots, \varnothing, a_i, \varnothing, \ldots, \varnothing \right\rangle$ are obtained by assigning each of its $|\mathcal{D}_{k+1}|$ actions to the first unassigned variable $p_{k+1}$.
- The leafs of the tree represent a (fully determined) joint move $\mathbf{a}_l$ (i.e. $\forall i : p_i \neq \varnothing$). In the tree, only leafs are assigned a value, which is equal to $\tilde{r}_{j\to i}(\mathbf{a}_l)$ (Equation 6.10).

The maximum value found in $\mathcal{T}(a_i)$ is the desired value. To speed up the search for this value, we can use branch and bound on this tree. In order to do this, we need to put bounds on the maximum value found in a subtree of $\mathcal{T}(a_i)$. These bounds depend on $U_j$ and the received messages $q$ (Equation 2.12). In many cases we can put bounds on the maximum of the former, that is obtained by further completing a partial joint action $\hat{\mathbf{a}}$ in a subtree of $\mathcal{T}(a_i)$. Combining these bounds on $U_j$ with with the minimum $\underline{q}(\hat{\mathbf{a}})$ and maximum $\overline{q}(\hat{\mathbf{a}})$ values of messages $q$ for $\hat{\mathbf{a}}$:

$$\underline{q}(\hat{\mathbf{a}}) = \sum_{\substack{k \in \mathcal{N}_j \setminus i \\ \hat{a}_k \neq \varnothing}} q_{k\to j}(\hat{a}_k) + \sum_{\substack{k \in \mathcal{N}_j \setminus i \\ \hat{a}_k = \varnothing}} \min_{p_k} q_{k\to j}(p_k)$$

$$\overline{q}(\hat{\mathbf{a}}) = \sum_{\substack{k \in \mathcal{N}_j \setminus i \\ \hat{a}_k \neq \varnothing}} q_{k\to j}(\hat{a}_k) + \sum_{\substack{k \in \mathcal{N}_j \setminus i \\ \hat{a}_k = \varnothing}} \max_{p_k} q_{k\to j}(p_k)$$

(where $\hat{a}_k$ is the $k^{\text{th}}$ element of $\hat{\mathbf{a}}$), gives us the desired bounds on $\tilde{r}_{j\to i}(\mathbf{p}_j)$ in Equation 6.10.

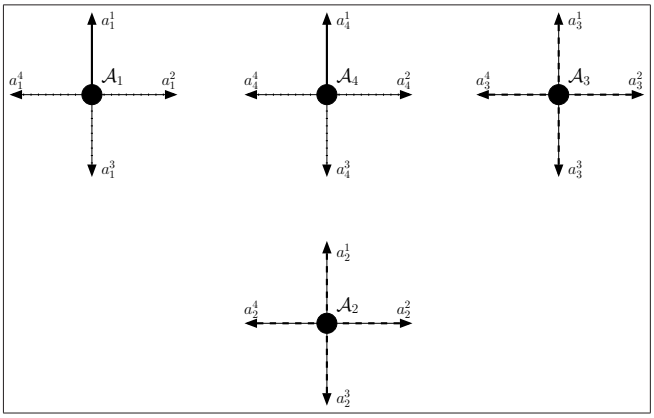
Figure 6.5 shows an example of a partially expanded search tree for computing a single element $r_{j\to 3}(a_3^1)$ of a message from function $U_j$ to variable $p_3$. Given this, the lower and upper bounds on the maximum (denoted between brackets) subtree $\left\langle a_1^1, \varnothing, a_3^1 \right\rangle$ can be pruned immediately after expanding the root. Similarly, subtree $\left\langle a_1^3, \varnothing, a_3^1 \right\rangle$ is pruned after expanding leaf $\left\langle a_1^2, a_2^2, a_3^1 \right\rangle$, which has the desired maximum value.

Since the utility functions $U_j$ are domain dependent in the context of the max-sum algorithm, there is no general way of computing these bounds. However, in most domains, such as the mobile agent domain, a partial joint action has a meaningful interpretation that can lead to an intuitive way of computing the bounds on $U_j$ in any subtree of $\mathcal{T}$: a partial joint action $\hat{\mathbf{a}}$ represents a situation in which only a subset of the agents have determined their action (Figure 6.6(a)). With this interpretation, we can obtain bounds as follows. The upper bound on this value is obtained by disregarding the agents that have not determined their action (i.e. agents $\mathcal{A}_i$ for which $p_i = \varnothing$). Since the act of making an observation always reduces the incremental value of other observations (because of the *submodularity* property of $f$ defined in Chapter 3), disregarding the observations of these 'undecided' agents will give an upper bound on the maximum (Figure 6.6(b)). To obtain a lower bound on the maximum, we use the *locality* property of the utility functions, which tells us that the interdependency between observations weakens as their distance increases. So, in order to obtain a lower bounds on the maximum, we chose the actions of the undecided agents that move them away from agent $\mathcal{A}_i$'s destination (Figure 6.6(c)).

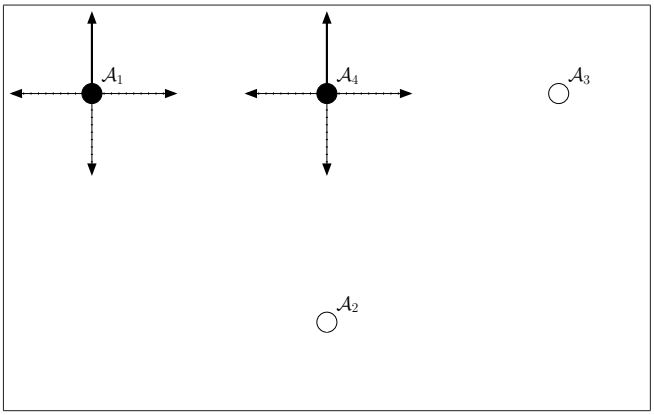
### 6.2.5 Ensuring Network Connectivity

Besides maximising observation value and a reduced computational overhead, it is also important in many situations that the agents maintain network connectivity, for example to transmit their measurements to a base station, or to coordinate their actions. Not surprisingly, we can use coordination to accomplish this, by penalising disconnection from the network in the utility function $U_j$. To this end, we assume that each agent maintains a routing table that specifies which agents can be reached through each immediate neighbour. Thus, an action is allowed if all agents are still be reachable through the links that remain after repositioning. Otherwise, the agent risks disconnection from the network, in which case a large penalty is added to its utility function $U_i$.

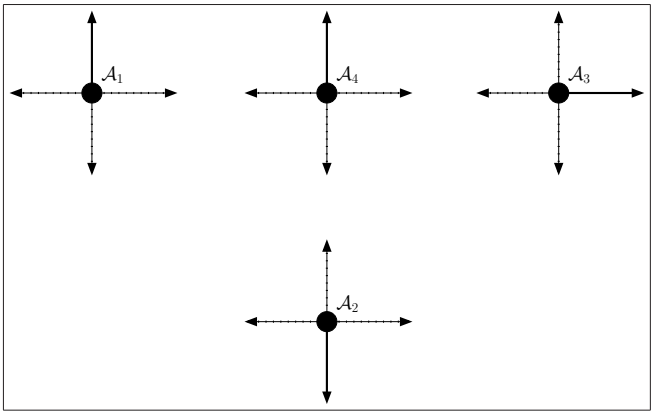
It is important to note, however, that although maintaining network connectivity is incentivised, the agents are not *guaranteed* to remain connected. This is due to the fact that max-sum is not guaranteed to converge in cyclic graphs (see Section 2.6.1), and theoretically, solutions can be arbitrarily bad when it does not (although empirical evidence suggests that, in practise, this does not happen often). So, the solution computed by max-sum can disconnect the network, even though this results in a strong negative utility. This problem can be solved by the use of the bounded version of max-sum (discussed in Section 2.6.1), which guarantees convergence and gives bounds on the computed solution, at the cost of a decrease in solution quality. By utilising this extension to the max-sum algorithm, it is possible to guarantee network connectivity by sacrificing a small amount of observation value. Here, however, we focus on the standard max-sum algorithm.




(a) A partial joint action $\hat{\mathbf{a}}\langle a_1^1, \varnothing, \varnothing, a_4^1, \rangle$, in which agents $\mathcal{A}_1$ and $\mathcal{A}_4$ move upwards, while agents $\mathcal{A}_2$ and $\mathcal{A}_3$ have not determined their actions yet. (Arrows indicate chosen actions; dotted arrows represent actions that have not been chosen; dashed arrows are actions that can still be chosen.)




(b) The upper bound of $\hat{\mathbf{a}}$ for $\mathcal{A}_2$ is computed by ignoring (the actions of) agents $\mathcal{A}_2$ and $\mathcal{A}_3$. In so doing, the value of the observations $\mathcal{A}_4$ collects is not decreased by the observations of $\mathcal{A}_2$ and $\mathcal{A}_3$. As a result, this scenario puts an upper bound on the maximum obtainable utility of $S_4$.




(c) A lower bound on $\hat{\mathbf{a}}$ for $\mathcal{A}_2$ is obtained by moving agents $\mathcal{A}_2$ and $\mathcal{A}_3$ *away* from the destination of $\mathcal{A}_4$, thereby minimising the dependence between the observations of $\mathcal{A}_2$ and $\mathcal{A}_3$ and the observations $\mathcal{A}_4$ collects.

FIGURE 6.6: Computing lower and upper bounds on the utility of a partial joint move.

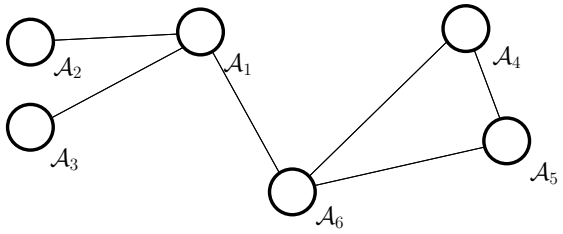



FIGURE 6.7: A communication network between agents.

Now, more formally, let $\mathcal{R}_i^{\mathcal{C}} : \mathbf{A} \to 2^{\mathbf{A}}$ be the routing table for agent $\mathcal{A}_i$ in the communication network represented by graph $\mathcal{C}$, that maps each agent to a set of agents that can be reached through it. If $\mathcal{A}_j$ is not a neighbour of $\mathcal{A}_i$ (i.e. no direct communication link exists), then $\mathcal{R}_i^{\mathcal{C}}(\mathcal{A}_j) = \emptyset$. Also, we define $\mathcal{R}_i^{\mathcal{C}}(\mathcal{A}_i) = \{\mathcal{A}_i\}$. Given this routing table, an agent can detect whether the communication network is connected by checking if all agents can be reached, i.e. $\bigcup_{j=1}^{M} \mathcal{R}_i^{\mathcal{C}}(\mathcal{A}_j) = \mathbf{A}$. When clear from the context, we will omit the superscript referring to the communication graph $\mathcal{C}$.

**Example 6.3** (Network Connectivity)**.** *Figure 6.7 shows a communication network among 6 agents. The routing table for agent $\mathcal{A}_6$ in this scenario is:*

$$\begin{aligned}
\mathcal{R}_6(\mathcal{A}_1) &= \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\} \\
\mathcal{R}_6(\mathcal{A}_2) &= \emptyset \\
\mathcal{R}_6(\mathcal{A}_3) &= \emptyset \\
\mathcal{R}_6(\mathcal{A}_4) &= \{\mathcal{A}_4, \mathcal{A}_5\} \\
\mathcal{R}_6(\mathcal{A}_5) &= \{\mathcal{A}_4, \mathcal{A}_5\} \\
\mathcal{R}_6(\mathcal{A}_6) &= \{\mathcal{A}_6\}
\end{aligned}$$

*Since $\{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\} \cup \{\mathcal{A}_4, \mathcal{A}_5\} \cup \{\mathcal{A}_6\} = \mathbf{A}$, the network is connected.*

Now, given the routing function $\mathcal{R}$, an agent is able to determine if a move will disconnect it from the network. In particular, if an action results in the disconnection of one of its neighbours $\mathcal{A}_j$, the network is still connected if $\bigcup_{i=1,i\neq j}^{M} \mathcal{R}(\mathcal{A}_i) = \mathbf{A}$. If, however, this condition does not hold, the agent risks disconnecting itself from (a part of) the communication network.

**Example 6.4** (Network Connectivity, Continued)**.** *Suppose agent $\mathcal{A}_6$ repositions itself, causing connection loss with agent $\mathcal{A}_4$. Now, $\bigcup_{i=1,i\neq 4}^{6} \mathcal{R}_6(i) = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6\} = \mathbf{A}$ so agent $\mathcal{A}_6$ is still connected to all other agents. Thus, any move by agent $\mathcal{A}_6$ that disconnects it from sensor $\mathcal{A}_4$* only*, does not disconnect the network. If, however, it loses connection with agent $\mathcal{A}_1$, $\bigcup_{i=1,i\neq 1}^{M} \mathcal{R}_6(\mathcal{A}_i) = \{\mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6\} \neq \mathbf{A}$, and agent $\mathcal{A}_6$ is no longer connected to agents $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$.*

In more detail, let $C(p_1, \ldots, p_M)$ be the communication network after the agents have moved according to their action variables $p_1, \ldots, p_M$. We can now define a *Network Disconnection Penalty* that penalises moves that disconnect the communication network:

**Definition 6.2** (Network Disconnection Penalty)**.**

$$P_i^{Conn}(p_1, \ldots, p_M) = \begin{cases} 0 & \text{if } \bigcup_{j=1}^{M} \mathcal{R}_i^{C(p_1,\ldots,p_M)}(\mathcal{A}_j) = \mathbf{A} \\ -\infty & \text{otherwise} \end{cases}$$

Of course, since only neighbours influence the result of $\bigcup_{j=1}^{M} \mathcal{R}_i(\mathcal{A}_j)$, only variables $p_j$ of the agents that are adjacent in the communication network need to be included in the set of parameters to $P_i^{Conn}$.

The penalty function is used by augmenting it to the original utility functions of the agents.

**Definition 6.3** (Augmented Agent Utility)**.**

$$U_i^{AUG}(\mathbf{p}_i) = U_i(\mathbf{p}_i) + P_i^{Conn}(\mathbf{p}_i)$$

In conjunction with max-sum, the augmented utility function incentivises the agents to avoid moves that result in the disconnection of an agent, since the team utility (see Equation 6.7) of such a move is $-\infty$.

More sophisticated forms of connectivity can also be incentivised with the same technique. In particular, *fault tolerance* is one of these more sophisticated forms. A communication network is said to be fault tolerant if it has at least two alternative paths between any pair of agents, making it more robust against failing agents. This coincides with the concept of 2-connected graphs, as established by Whitney's Theorem (Gross & Yellen 1999). In more detail, 2-Connectivity implies that at least two agents need to fail simultaneously before the network becomes disconnected (or is reduced to a single agent). Clearly, the communication network from Example 6.3 is not 2-connected; the failure of agent $\mathcal{A}_6$ breaks the network into two components: $\{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}$ and $\{\mathcal{A}_4, \mathcal{A}_5\}$

By inspecting its routing table, an agent can easily determine if it satisfies the necessary condition for 2-connectivity; its failure should still leave the communication network connected. In order to do this, Algorithm 9 counts the number of components the network is divided into should the agent fail; if this number is equal to 1, the network is fault tolerant.

Now, the following penalty function can be used to augment the utility function of the agent in order to enforce fault tolerance in a decentralised fashion.

**Algorithm 9** A decentralised algorithm for determining whether a communication network is fault tolerant.

**Require:** $\mathcal{R}_i^{\mathcal{C}}$: routing table for agent $\mathcal{A}_i$
**Ensure:** Returns **true** iff failure of agent $\mathcal{A}_i$ does not disconnect the communication network
*Initialisation:*
**for all** $\mathcal{A}_j \in \mathbf{A} \setminus \{\mathcal{A}_i\}$ **do**
  $\mathbf{C} \leftarrow \mathbf{C} \cup \mathcal{R}_i(\mathcal{A}_j)$
**end for**
*Closure:*
**while** $\exists C_1, C_2 \in \mathbf{C} : C_1 \cap C_2 \neq \emptyset$ **do**
  $\mathbf{C} \leftarrow \mathbf{C} \setminus \{C_1, C_2\}$
  $\mathbf{C} \leftarrow \mathbf{C} \cup \{C_1 \cup C_2\}$
**end while**
**return** **true** if $|\mathbf{C}| = 1$, **false** otherwise

**Definition 6.4** (2-Connectedness Violation Penalty)**.**

$$P_i^{2\text{-}Conn}(p_1, \ldots, p_M) = \begin{cases} 0 & \text{if } isFaultTolerant(i, C(p_1, \ldots, p_M)) \\ -\infty & \text{otherwise} \end{cases} \tag{6.11}$$

In general, the technique of augmenting the utility function with a penalty function is applicable to enforce various constraints on the agents' collective movement. The only requirement on such a constraint is that its violation should be locally detectable. That is, one or more agents should be able to determine if the constraint is satisfied based on information received from direct neighbours or from its own observations. If this is the case, a penalty function can be effectively used to filter out solutions that do not satisfy the constraint. Other examples, besides the ones given above, include the assignment of roles to agents (i.e. some agents function as communication relays, others as explorers, etc.), and to ensure certain events or locations are monitored by agents.

## 6.3 Empirical Evaluation

In this section we empirically evaluate the algorithm developed in the previous section within three different information gathering domains:[9]

**Monitoring Environmental Phenomena** In this scenario, the agents are tasked with monitoring an environmental phenomenon, such as temperature (Section 6.3.1).

**Pursuit Evasion** In this scenario, the agents are tasked with capturing a moving object (Section 6.3.2).

[9]Empirical results on the first domain were published in Stranders et al. (2009*b*). Results for the second and third domain were published in Stranders, Delle Fave, Rogers & Jennings (2010).

**Patrolling** In this scenario, the agents' goal is to prevent attackers from intruding their environment (Section 6.3.3).

The structure of these sections is identical. First, we derive a value of information, by showing how the environment can be represented. Then, we discuss the experimental setup, and the benchmark strategies against which we compared our algorithm. Finally, we present and analyse the results.

To allow for a dynamic view of the operation of our algorithm, we have made videos available of all these experiments, which can be found at `http://users.ecs.soton.ac.uk/rs06r/videos/`.

### 6.3.1 Domain 1: Monitoring Environmental Phenomena

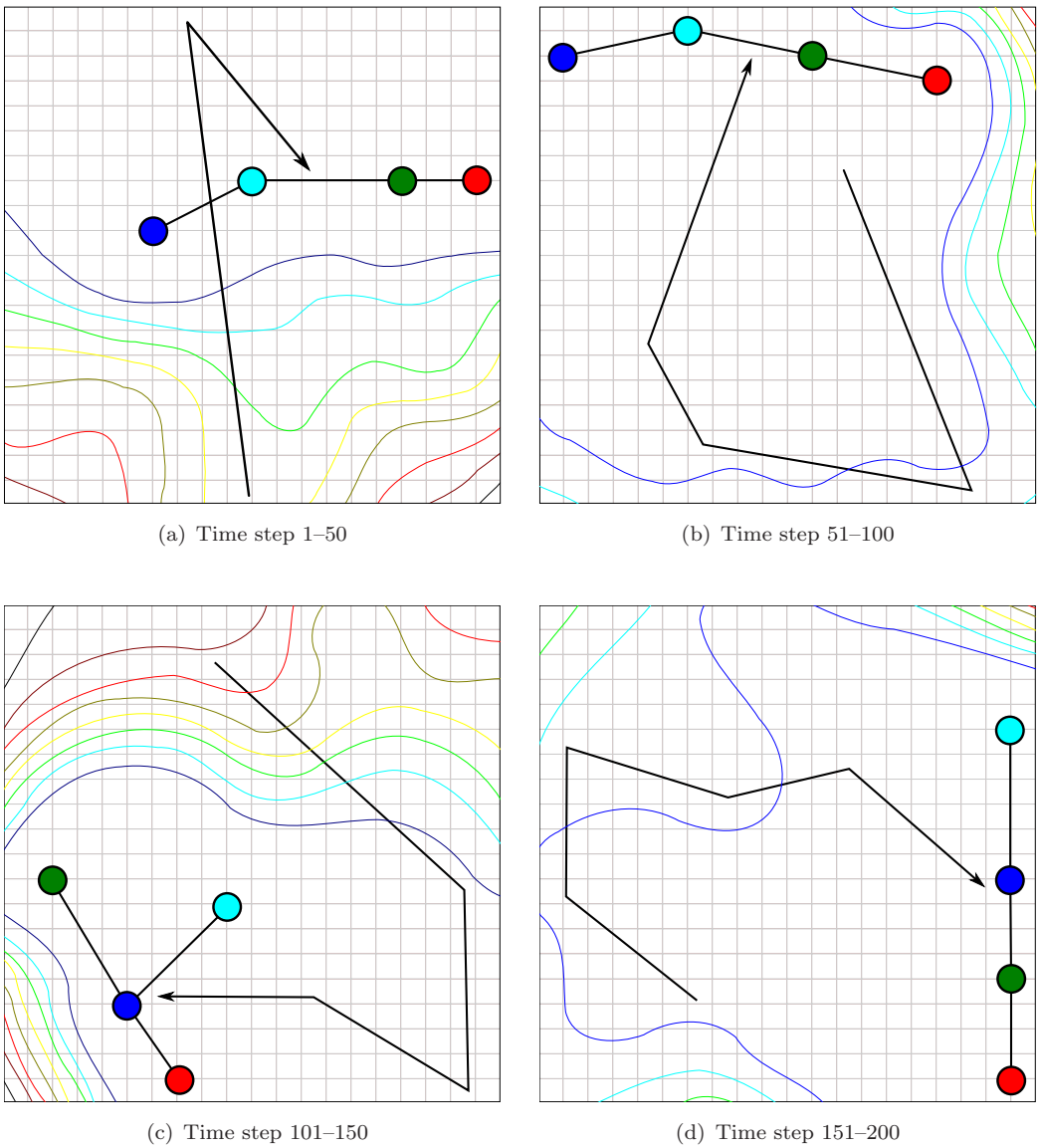
The first domain we use for empirically evaluating the algorithm involves agents monitoring an environmental phenomenon (in this case, temperature) in a rectangular environment. The agents' goal is to observe the phenomenon (i.e. take temperature measurements) in such a way as to provide accurate situational awareness. More concretely, this means that agents need to coordinate their movements to take those observations that enable them to predict the value of the phenomenon at unobserved locations as accurately as possible.

Figure 6.8 shows four snapshots of this scenario, taken at time steps 50, 100, 150 and 200. Note how the team makes a sweeping motion through its environment, in order to maintain low uncertainty throughout, and thus minimise the predictive variance with which the environmental phenomenon can be predicted. It is also worth noting that the team maintains connectivity throughout the simulation, using techniques discussed in Section 6.2.5.

Before applying the coordination algorithm in this setting, we first choose an appropriate metric for valuing observations.

#### 6.3.1.1 Valuing Observations

In Section 2.3.1.2, we argued that the GP is a flexible and effective regression technique for modelling environmental phenomena. For this reason we will use it in this domain to model the agents' environment and predict the value of the phenomenon at unobserved locations. In terms of valuing observations, recall from Section 2.4 that within the context of the GP, we have a choice between entropy and mutual information. The former is a *local* metric, and is comparatively cheap to evaluate, but it sometimes causes agents to waste their effective sensing range by moving close to the borders of the environment. The latter demands more computational resources, but avoids this undesirable effect.



(a) Time step 1–50

(b) Time step 51–100

(c) Time step 101–150

(d) Time step 151–200

FIGURE 6.8: Snapshots at time steps 50, 100, 150 and 200 of a simulation of four agents monitoring an environmental phenomenon. The grid represents graph $G$, which allows the agents to move in four directions. The arrows indicate the approximate path of the team in the 50 time steps between two subsequent snapshots. The lines between the agents indicate that communication between them is possible.

Thus, this choice constitutes a trade-off between solution quality and computational overhead.

To make this trade-off in an informed manner, we compared these two metrics using the same parameters and physical layout we will use to benchmark the coordination algorithm later. Initially, it was our aim to use the algorithm itself to make this comparison, but despite the various improvements to lessen the computational demand of the max-sum algorithm, repeatedly evaluating mutual information for a large set of observations proved too demanding. Instead, we chose to use an un-negotiated (see Section 2.6)

greedy algorithm[10] in two settings: deploying fixed agents, and controlling the movement of mobile agents. We recorded the algorithm's performance in terms of solution quality and computation time for a varying number of agents. Specifically, solution quality is expressed as the average root mean squared error (RMSE) over all time steps, which is calculated as follows:

$$\frac{1}{|T|}\sum_{t\in T}\sqrt{\sum_{v\in V}\frac{\sigma^2_{v,t}}{|V|}} \tag{6.12}$$

where $\sigma^2_{v,t}$ is the variance at spatio-temporal coordinates $(v,t)$, which is calculated by Equation 2.2.

The aforementioned difference in behaviour of these two metrics is clearly visible in Figures 6.9 and Figure 6.10 for fixed and mobile agents respectively; while maximising entropy the agents tend to venture close to the edge of the environment than when maximising mutual information. These findings are consistent with those of Guestrin et al. (2005) (see Section 2.4).

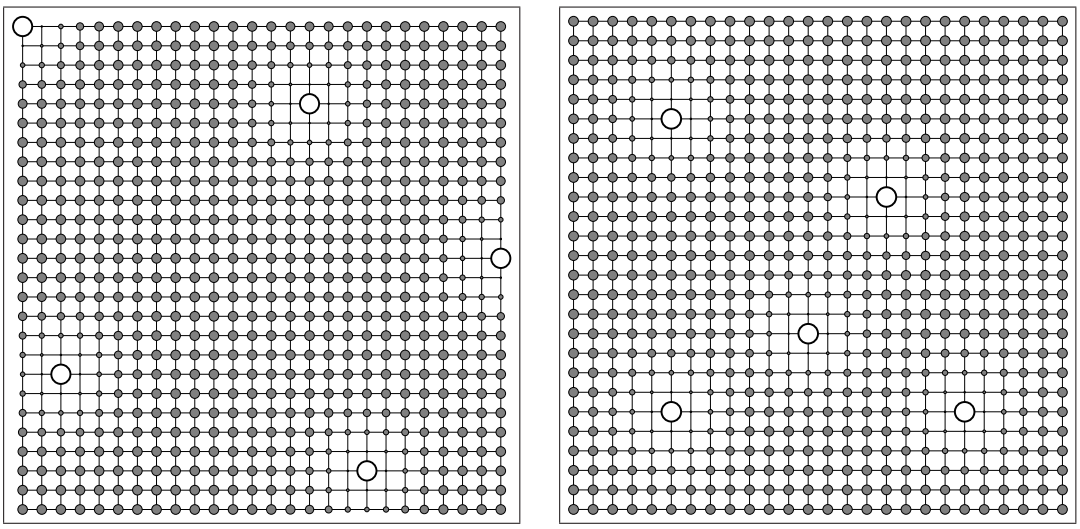
However, if we analyse this difference in terms of the average RMSE over 100 time steps (Figure 6.11(a)), we find that mutual information provides a decrease of RMSE of only 10%. Moreover, this decrease comes at a significant cost (Figure 6.11(b)). The repeated evaluation of mutual information required by the greedy algorithm requires roughly two orders of magnitude more computation time than entropy.[11] We deem this too high a price to pay for a small increase in solution quality, especially considering that our algorithm needs to compute the value of many different sets of observations at every time coordination between the agents is scheduled.

In light of this, we opt for the entropy metric in this domain. Using this metric, the utility function of an individual agent (Definition 6.1) is defined as:

$$\begin{aligned} U_i(O_1,\ldots,O_i) &= \rho_{O_i}\left(\bigcup_{j=1}^{i-1} O_j \cup \mathbf{O}^t_{\mathbf{A}}\right) \\ &= f(O_1\cup\ldots\cup O_i\cup\mathbf{O}^{t-1}_{\mathbf{A}}) - f(O_1\cup\ldots\cup O_{i-1}\cup\mathbf{O}^{t-1}_{\mathbf{A}}) \\ &= H(O_1\cup\ldots\cup O_i\cup\mathbf{O}^{t-1}_{\mathbf{A}}) - H(O_1\cup\ldots\cup O_{i-1}\cup\mathbf{O}^{t-1}_{\mathbf{A}}) \\ &= H(O_i \mid O_1\cup\ldots\cup O_{i-1}\cup\mathbf{O}^{t-1}_{\mathbf{A}}) \end{aligned} \tag{6.13}$$

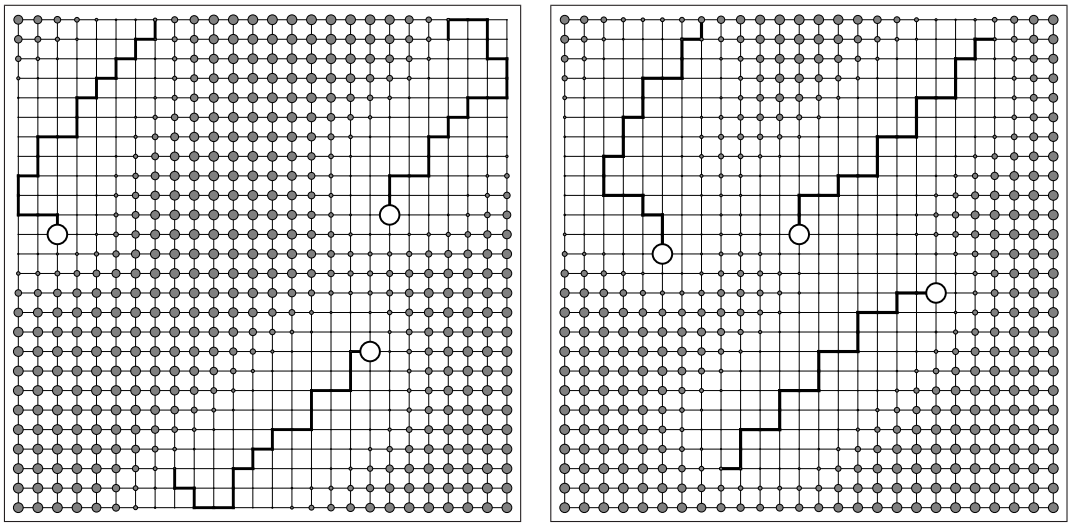
[10] This algorithm was part of our initial study into coordinating mobile agents, and was published as Stranders et al. (2008)

[11] These results were obtained after optimising the computation of mutual information by exploiting the locality property of $f$. That is, the entropy reduction that results from making an observation is computed only for those locations within a range of $\delta$. In this case, this range was $\delta = 20$, with an error of $\rho < 0.01$.



(a) Entropy (b) Mutual Information

FIGURE 6.9: Two deployments of fixed agents using the entropy and mutual information value metrics. The size of the vertices is proportional to the RMSE at those locations.



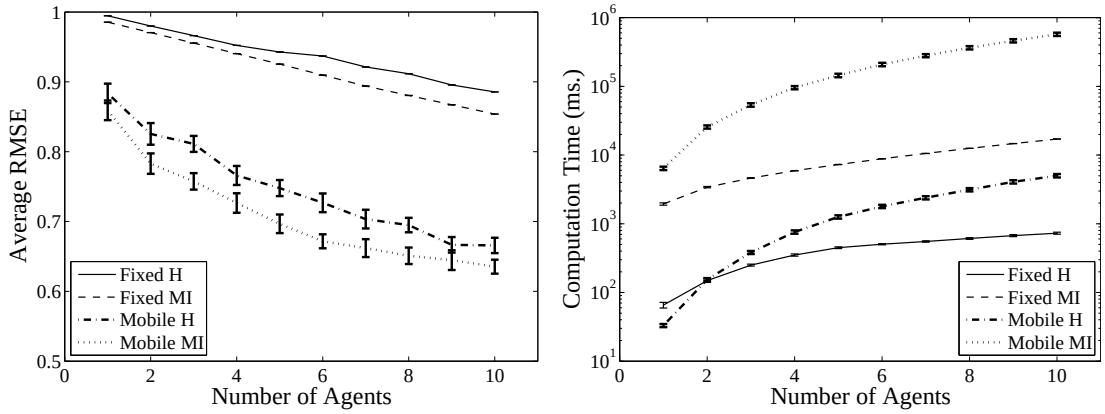
(a) Entropy (b) Mutual Information

FIGURE 6.10: Two snapshots at time step 20 of mobile agents using the entropy and mutual information value metrics. The size of the vertices is proportional to the RMSE at those locations.

In other words, the utility of agent $\mathcal{A}_i$ is the entropy of observations $O_i$ conditioned on $O_1 \ldots \cup O_{i-1} \cup \mathbf{O}_{\mathbf{A}}^{t-1}$, or, equivalently, the entropy of $O_i$ that remains after the agents with a lower index have made their observations.

#### 6.3.1.2 Experimental Setup

To empirically evaluate the algorithm developed in this chapter in this domain, we simulated five agents on a lattice graph measuring 26 by 26 vertices (see for example




(a) (b)

FIGURE 6.11: A comparison of the entropy (H) and mutual information (MI) value metrics. Errorbars indicate the standard error in the mean.

Figure 6.11(a)). The environmental phenomenon was generated by a GP with a squared exponential covariance function (see Equation 2.3) with a spatial length-scale of 10 and a temporal length-scale of 150. This yields an environmental phenomenon with a strong correlation along the temporal dimension, which therefore changes slowly over time. At every $m$ time steps, the agents plan their motion for the next $l$ time steps ($l \geq m$). In what follows, this strategy is referred to as MS$m$-$l$.

Since the environmental layout is very structured (i.e. a lattice), we use heuristic 1 to determine the action space of the agents (see Section 6.2.2). Thus, each agent considers paths of length $l$ in 8 different directions. We will compare the two action selection heuristics in the next two experiments, where we use a layout with obstacles.

In this experiment, we benchmarked MS1-1 and MS1-5 against four strategies:

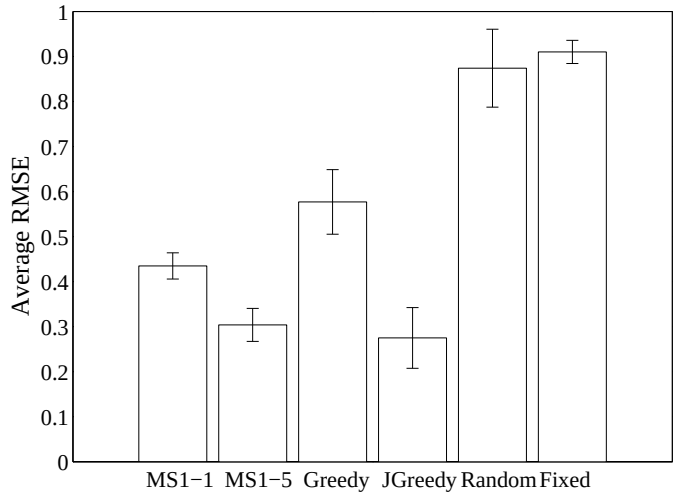
**Random** Randomly moving agents.

**Greedy** These agents use the un-negotiated greedy coordination algorithm that moves to the adjacent location with the highest observation value.

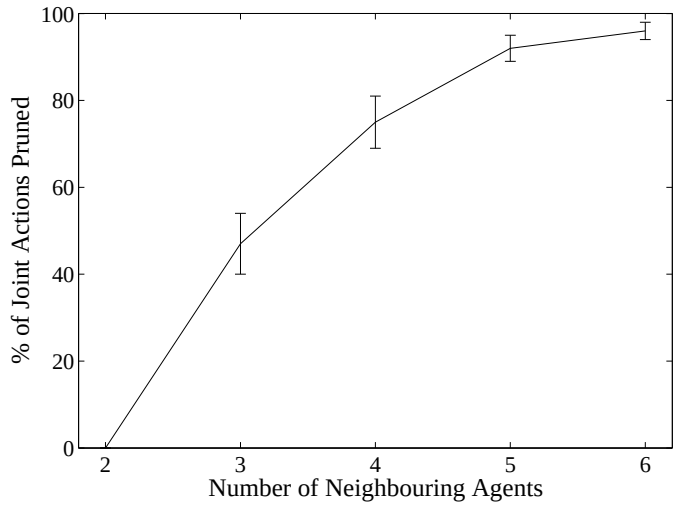
**J(umping) Greedy** The same as Greedy, except that these agents can instantaneously jump to any location.

**Fixed** Fixed agents that are placed using a greedy algorithm for maximising mutual information (Guestrin et al. 2005), which we used earlier to compare entropy with mutual information.

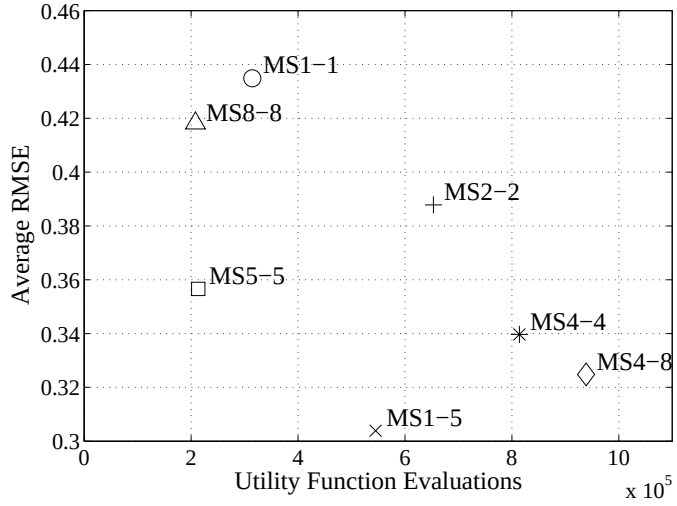
We randomly generated 200 problem instances (i.e. initial positions for the agents), and evaluated the performance of all algorithms on each of these instances.




(a)




(b)




(c)

FIGURE 6.12: Empirical performance of the MS1-1 and MS1-5 algorithms for monitoring environmental phenomena. Errorbars indicate the standard error in the mean.

#### 6.3.1.3 Results

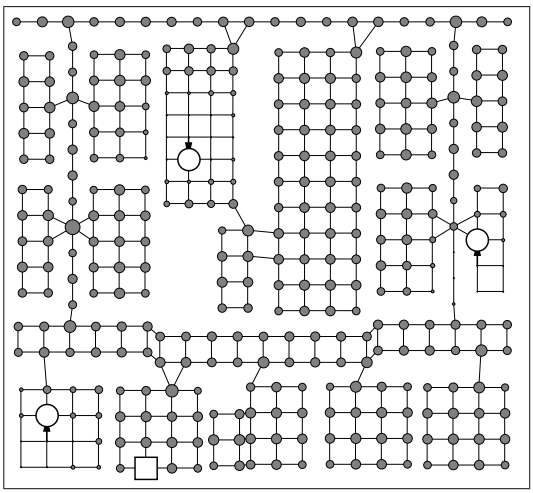
The averaged RMSE over 100 time steps is plotted in Figure 6.12(a). From this, it is clear that both MS strategies outperform the Greedy, Random, and Fixed strategies. Furthermore, the prediction accuracy of MS1-5 is comparable to that of JGreedy, whose movement is not restricted by graph $G$. Moreover, it shows that increasing the length of the look ahead from 1 to 5, reduces the RMSE by approximately 30%.

We also analysed the speed-up achieved by applying the two pruning techniques described in Section 6.2.4.1. Figure 6.12(b) shows the percentage of joint actions pruned plotted against the number of neighbouring agents. If an agent has 5 neighbours, the two pruning techniques combined prune around 92% of the joint moves. With such a large number of neighbours, the agents are strongly clustered, which occurs rarely in large environments. Should this nevertheless happen, the agent in question needs to evaluate its utility function for only 8% of roughly $8^5$ joint actions, thus greatly improving the algorithm's efficiency.

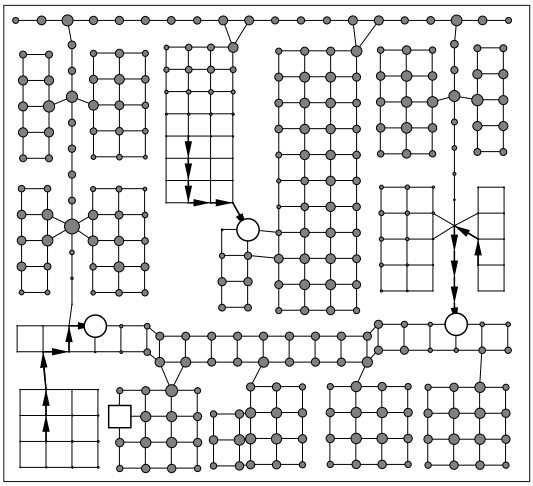
Finally, we performed a cost/benefit analysis of various MS$m$-$l$ strategies. More specifically, we examined the effect of varying $m$ and $l$ on both the number of utility function evaluations, and the resulting RMSE. Figure 6.12(c) shows the results. The behaviours of MS1-1, MS2-2, MS4-4, MS5-5, and MS8-8 show an interesting pattern. Up to and including $m = l = 4$, both the number of function evaluations and the average RMSE decrease. This is due to the fact that planning longer paths is more expensive, but results in lower RMSE. However, for $m, l > 4$, the action space becomes too coarse (since only 8 directions are considered) to maintain a low RMSE. At the same time, the number of times the agents coordinate reduces significantly, resulting in a lower number of function evaluations. MS1-5 and MS4-8 provide a compromise. They compute longer paths, but coordinate more frequently. This leads to more computation compared to MS5-5 and MS8-8, but results in significantly lower RMSE, because agents are able to 'reconsider' their paths.

### 6.3.2 Domain 2: Pursuit Evasion

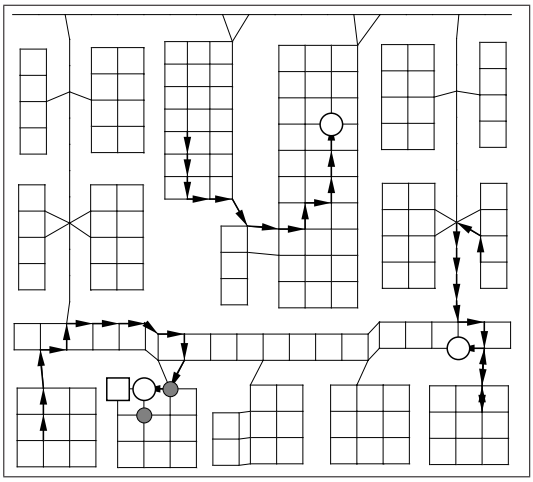
Pursuit evasion is the second domain in which we evaluate our algorithm. This domain is characterised by the presence of a single moving object $e$ (called an evader) that the agents need to capture as quickly as possible. The evader $e$ has a type $M$ (e.g. random, stationary, etc.) that specifies how it moves in graph $G$. We assume that agents are capable of sensing all locations within a radius of $r_s$ of their position, but imperfectly: agents make false positive or false negative observations with probabilities $p_{fp}$ and $p_{fn}$. Finally, an evader is captured when it is within one of the agents' capture ranges $r_c$. Figure 6.13 shows three snapshots of a pursuit evasion scenario with a randomly moving evader.



(a) $t = 1$



(b) $t = 6$



(c) $t = 13$

FIGURE 6.13: An example pursuit-evasion scenario with three agents. The big circles represent the agents and the square represents the evader. The size of the grey circles is proportional to the observation value $f$ at that location. The evader is captured at $t = 14$.

#### 6.3.2.1 Valuing Observations

As in Section 6.3.1, we start with deriving an observation value function. This function assigns a value to observations which is proportional to the probability that the evader will be detected. To do this, the agents model their belief of the evader's location $e_t$ at time $t$ using a probabilistic map $p_e(e_t = v \mid \mathbf{O}_\mathbf{A}^{t-1})$, representing the probability that the evader is at location $v$ given their observation history $\mathbf{O}_\mathbf{A}^{t-1}$. This model extends the work by Hespanha et al. (1999) to a setting with an arbitrary evader (as opposed to a randomly moving one), or an unknown evader. Each agent has a copy of this probabilistic map which is kept consistent by exchanging observations. At each time step $t$, the agents take new measurements $O_\mathbf{A}^t$ and obtain $p_e(e_{t+1} = v \mid \mathbf{O}_\mathbf{A}^t)$ in two steps:

1. Fuse measurements $O_\mathbf{A}^t$ with $p_e(e_t = v \mid \mathbf{O}_\mathbf{A}^{t-1})$ to obtain:

$$p_e(e_t = v \mid \mathbf{O}_\mathbf{A}^t) = \alpha \cdot p_e(e_t = v \mid \mathbf{O}_\mathbf{A}^{t-1}) \cdot p(O_\mathbf{A}^t \mid e_t = v, \mathbf{O}_\mathbf{A}^{t-1})$$

   Here, $\alpha$ is a normalising constant, and $p(O_\mathbf{A}^t \mid e_t = v, \mathbf{O}_\mathbf{A}^{t-1})$ is computed as:

$$p(O_\mathbf{A}^t \mid e_t = v, \mathbf{O}_\mathbf{A}^{t-1}) = \prod_{o \in O_\mathbf{A}^t} p(o \mid e_t = v, \mathbf{O}_\mathbf{A}^{t-1})$$

   and $p(o \mid e_t = v, \mathbf{O}_\mathbf{A}^{t-1})$ is given by probabilities of false negatives and false positives:

$$p(o \mid e_t = v, \mathbf{O}_\mathbf{A}^{t-1}) = \begin{cases} 1 - p_{fp} & \text{if } o(m) = T \wedge o(l) = v \\ p_{fp} & \text{if } o(m) = T \wedge o(l) \neq v \\ p_{fn} & \text{if } o(m) = F \wedge o(l) = v \\ 1 - p_{fn} & \text{if } o(m) = F \wedge o(l) \neq v \end{cases}$$

   Recall from Chapter 3, that $o(m)$ is the realisation of observation $o$, in this case whether an evader was detected, and $o(l)$ is its spatial coordinate.

2. Predict the motion of the evader. If the evader's type $M$ is known, the agents compute:

$$p_e(e_{t+1} = v \mid \mathbf{O}_\mathbf{A}^t) = \sum_{v' \in V} p_m(e_{t+1} = v \mid e_t = v', M) p_e(e_t = v' \mid \mathbf{O}_\mathbf{A}^t) \qquad (6.14)$$

   Here, $p_m(e_{t+1} = v \mid e_t = v', M)$ is the transition (movement) probability of evader type $M$ from $v$ to $v'$. Since the evader's movement is restricted by layout graph $G(V, E)$, $p_m(e_{t+1} = v \mid e_t = v', M) = 0$ if $(v, v') \notin E$. If, however, the evader's type $M$ is unknown, but it is known that $M \in \{M_1, \ldots, M_n\}$, the agents compute

a posterior over types $M$ given observations $\mathbf{O}_{\mathbf{A}}^{t-1}$:

$$p(M_i \mid \mathbf{O}_{\mathbf{A}}^{t}) = \alpha \cdot p(O_{\mathbf{A}}^{t} \mid \mathbf{O}_{\mathbf{A}}^{t-1}, M_i) \cdot p(M_i \mid \mathbf{O}_{\mathbf{A}}^{t-1}) \tag{6.15}$$

where $p(O_{\mathbf{A}}^{t} \mid \mathbf{O}_{\mathbf{A}}^{t-1}, M_i) =$

$$\sum_{v \in V} p(O_{\mathbf{A}}^{t} \mid e_t = v, \mathbf{O}_{\mathbf{A}}^{t-1}, M_i) \cdot p_e(e_t = v \mid \mathbf{O}_{\mathbf{A}}^{t-1}, M_i) \tag{6.16}$$

In this case, $p_e(e_{t+1} = v \mid \mathbf{O}_{\mathbf{A}}^{t})$ is computed as:

$$\sum_{i=1}^{n} p_e(e_{t+1} = v \mid \mathbf{O}_{\mathbf{A}}^{t}, M_i)$$

Using the probability map obtained from this computation, the measurement value function $f$ is defined as:

$$f(O_{\mathbf{A}}^{t} \cup \cdots \cup O_{\mathbf{A}}^{t+l} \cup \mathbf{O}_{\mathbf{A}}^{t-1}) = 1 - \prod_{t'=t}^{t+l} \left( 1 - \sum_{o \in O_{\mathbf{A}}^{t'}} p_e(e_{t+1} = o(l) \mid \mathbf{O}_{\mathbf{A}}^{t-1}) \right)$$

which is the probability that at least one observation made by the agents collectively in the next $l$ time steps will result in the detection of the evader. Once an agent has made a positive observation, the agents suspend their information gathering tasks and move to the location where the evader is expected to be in the next time step to capture it.

#### 6.3.2.2 Experimental Setup

To evaluate our algorithm, we consider two different graphs:

**Office** The layout of this environment is a model of the IAM lab at the University of Southampton, which measures 67 by 47.5 metres (see Figure 6.13). The sensing range $r_s = 9$m and capture range $r_c = 4$m. The number of agents $M = 4$.

**Lattice** A 26 by 26 square lattice graph measuring 100 by 100 metres, which was used in the previous experiment (see Figure 6.10). Sensing range $r_s = 10$m, and capture range $r_c = 4$m. The number of agents $M = 5$.

These two types of graphs were chosen to illustrate the effect of the two action selection heuristics discussed in Section 6.2.2 on the performance of our algorithm compared in a structured and less-structured graph.

We used three different types of evaders: stationary, random (which moves to a random adjacent location) or smart (which moves away from the closest agent), with equal

probability. The first two can be used to model civilians in a disaster scenario; the second an intruder in a security domain.

For all problem instances, the probability of a false positive or false negative observation are $p_{fp} = p_{fn} = 0.001$.[12] The key metric we used to measure the agents' performance is the *capture time*; the time needed by the agent to capture the evader. We benchmarked our algorithm against the state of the art, as well as algorithms that provide an upper (JGreedy) and a lower bound (GRandom) on achievable performance:

**(G)Greedy** Greedy agents are controlled by the un-negotiated coordination algorithm we used before, which moves the agent to the adjacent location with the highest value in the next step. GGreedy is similar to Greedy, but moves agents toward the *global* location with highest value. These algorithms are state of the art for decentralised control in pursuit-evasion, and were proposed by Vidal et al. (2001).

**JGreedy** This approach instantaneously jumps to the global location with the highest value, as in Section 6.3.1.2.

**(G)Random** These are two random approaches. Random agents move to a random location adjacent to the agent's current position. GRandom selects a random position in the graph and then moves along the shortest path.

**TSP** This is a state of the art approach proposed by Sak et al. (2008). It computes the shortest closed walk that visits all vertices (similar to the Travelling Salesman Problem[13]). To improve its adaptiveness and competitiveness, we let the agents deviate from this walk once an evader is detected.

**MS-8** This is our algorithm, with action selection heuristic 1, recomputation interval $m = 5$ and look ahead $l = 15$. These values are identical to the configuration of MS K-M, which allows us to compare heuristic 1 and 2 under the same conditions.

**MS K-M** This is algorithm with action selection heuristic 2. Here, $m = 5$, $l = 15$, and the number of clusters $c = 4$.[14] These values were chosen after initial calibration of the algorithm on both layout graphs.

We randomly generated 200 problem instances, and evaluated the performance of all algorithms on each of these instances.

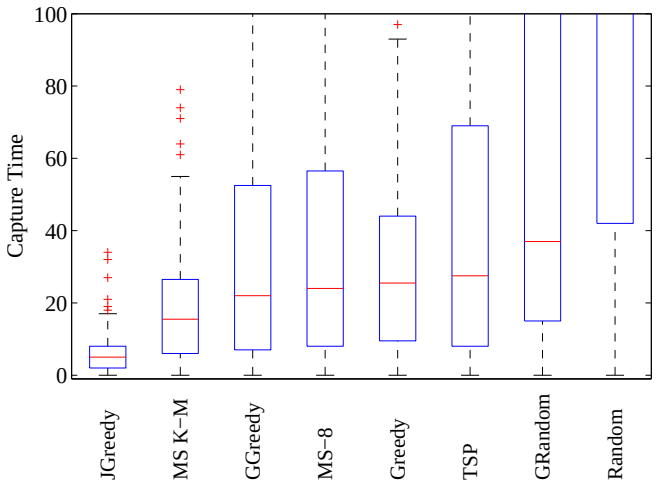



FIGURE 6.14: Empirical performance on 200 instances of the pursuit-evasion domain. The band near the centre of the boxes is the median of the dataset; the box contains data points between the $25^{th}$ and $75^{th}$ percentile. Whiskers are drawn at 1.5 inter-quartile range (IQR).

#### 6.3.2.3 Results

The results for the office environment are summarised in box-plots shown in Figure 6.14, showing that MS K-M has a lower median compared to GGreedy, MS-8, Greedy and TSP, as well as a significantly lower variance. From this, we can conclude that this algorithm consistently and reliably captures the evader sooner than the benchmark algorithms. Moreover, these results clearly demonstrate the superiority of heuristic 2 over heuristic 1 for selecting the paths over which the agents coordinate in the office environment, which is more constrained than the lattice graph. The explanation for this lies in the fact that heuristic 2 uses knowledge about the local topology of the graph and the current observation value associated with each vertex, instead of considering paths in 8 fixed directions like heuristic 1.

To determine the quantitative performance gain of MS K-M, we also performed a paired Student's t-test to compute 95% confidence intervals on the difference in performance between our algorithm and the benchmark algorithms. Table 6.3 reports the lower bounds of these confidence intervals (with the upper bound being $\infty$). Again, it is clear that the adaptive clustering heuristic results in a significant improvement ($> 40\%$) over MS-8 in the office environment. In the more structured lattice graph this improvement is more moderate (but still significant). Most importantly, our algorithm with heuristic 2 outperforms all benchmarks by at least 30% in the office environment.

[12]For a problem instance lasting for 100 time steps with 4 agents taking approximately 15 measurements per time step, the expected number of false positives is 6.

[13]To compute the TSP cycle of the graph, we used Concorde. (`http://www.tsp.gatech.edu/concorde.html`).

[14]Since both graphs have an average degree of equal to 4, in most cases, these clusters are reached through all outgoing edges.

| Environment | % Improvement (95% CI) | | | |
|---|---|---|---|---|
| | **Greedy** | **GGreedy** | **MS-8** | **TSP** |
| Office | 30.5% | 39.5% | 42.2% | 52.2% |
| Lattice | 26.9% | 42.0% | 10.0% | 47.4% |

TABLE 6.3: Lower bounds of the 95% confidence intervals of performance increase of MS K-M compared to the four most competitive benchmarks in the pursuit evasion domain.

### 6.3.3 Domain 3: Patrolling

In the patrolling domain, the third and final domain we consider in this chapter, the agents' goal is to prevent attacks on vertices of graph $G$. An attacker can start an attack on a vertex at any time (if no attacker is already present) and is successful if it is not captured by an agent within $k$ time steps. This models intruders that require time to break into the environment and steal items of value. Each time an attack is successful, the agents incur a loss of 1. This domain can be regarded as a variation of the pursuit-evasion domain, but instead of having a single moving evader that is already present in the environment, there are now multiple stationary attackers that can appear at any time. Apart from this, the domains the domains are identical. Observations are imperfect, agents have a observation range of $r_s$ and can capture attackers within a range of $r_c$.

#### 6.3.3.1 Valuing Observations

Similar to the pursuit-evasion domain, the agents maintain a probability map representing the probability that an attacker is present at each vertex of the layout graph. However, for every vertex, they not only track whether an attacker is present, but also for how long. This enables them to prioritise the attacks based on how far they have progressed, and interrupt those attacks that will otherwise succeed in the very near future.

To keep track of the state of a possible attack at a vertex $v$, we build a Markov model with $k+1$ states $S = \{\varnothing, 1, \ldots, k\}$, where $\varnothing$ indicates that no attacker is present, and $1, \ldots, k$ represents that an attacker has been present for the indicated amount of time. Valid state transitions are:

- $\varnothing \to 1$: an attacker appears.
- $i \to (i+1)$ for $1 \leq i \leq (k-1)$: the attack progresses one time step.
- $i \to \varnothing$ for $1 \leq i \leq k$: the attack is interrupted by an agent. No loss is incurred.
- $k \to \varnothing$: the attack succeeds and the agents incur a loss of 1.

Using this Markov model, we can now construct a probability map $p_a(s_t^v \mid \mathbf{O}_\mathbf{A}^{t-1})$ representing the probability that a vertex $v$ is in state $s_t^v \in S$ at time $t$ conditioned on previously made observations. To obtain $p_a(s_{t+1}^v \mid \mathbf{O}_\mathbf{A}^t)$ from new measurements $O_\mathbf{A}^t$ the agents follow two step computation, which is similar to the computation performed in pursuit-evasion (Section 6.3.2.1):

1. Fuse measurements $O_\mathbf{A}^t$ with $p_a(s_t^v \mid \mathbf{O}_\mathbf{A}^{t-1})$ to obtain $p_a(s_t^v \mid \mathbf{O}_\mathbf{A}^t)$. For ease of exposition, let us first assume perfect sensing (i.e. $p_{fn} = 0$ and $p_{fp} = 0$). Under this assumption, two cases can be distinguished: $v$ is currently under attack (an event we denote by $a$) or it is not (denoted by $\neg a$). In the former case, we clearly have that $p_a(s_{t+1}^v = \varnothing \mid a) = 0$. For $i \neq \varnothing$ we calculate:

$$p_a(s_{t+1}^v = i \mid a) = \frac{p_a(s_t^v = i \mid \mathbf{O}_\mathbf{A}^{t-1})}{1 - p_a(s_v^t = \varnothing \mid \mathbf{O}_\mathbf{A}^{t-1})}$$

   In the latter case, we clearly have $p_a(s_{t+1}^v = \varnothing \mid \neg a) = 1$.

   Now, dropping the assumption of perfect sensing, note that in case of a negative observation an attacker might still be present with probability $p_{fn}$. Thus, to obtain $p_a(s_{t+1}^v \mid \mathbf{O}_\mathbf{A}^t)$, we weigh vectors $p_a(s_{t+1}^v \mid a)$ by $p_{fn}$ and $p_a(s_{t+1}^v \mid \neg a)$ by $1 - p_{fn}$. The converse holds for a positive observation.

2. Update the probability map taking into account possible new attacks. If we suppose that at every time step $t \in T$ and every $v \in V$ the probability $p$ of a new attacker appearing is constant and independent, this Markov model is fully defined by the following probabilistic transition function:

$$p_t(s_{t+1}^v \mid \mathbf{O}_\mathbf{A}^t, s_t^v) = \begin{cases} 1-p & \text{if } s_{t+1}^v = \varnothing \text{ and } s_t^v = \varnothing \\ p & \text{if } s_{t+1}^v = 1 \text{ and } s_t^v = \varnothing \\ 1 & \text{if } s_{t+1}^v = i \text{ and } s_t^v = i-1 \text{ for } i \leq k-1 \\ 1 & \text{if } s_{t+1}^v = \varnothing \text{ and } s_t^v = 1 \\ 0 & \text{otherwise} \end{cases}$$

   Using this function we compute:

$$p_a(s_{t+1}^v \mid \mathbf{O}_\mathbf{A}^t) = p_t(s_{t+1}^v \mid \mathbf{O}_\mathbf{A}^t, s_t^v) \cdot p_a(s_t^v \mid \mathbf{O}_\mathbf{A}^t)$$

   to obtain the updated probability map.

After performing this computation, we can define observation value function $f$ for this domain in terms of the probability of an attack currently in progress at the observed

locations as follows:

$$f(O_{\mathbf{A}}^{t} \cup \cdots \cup O_{\mathbf{A}}^{t+l} \cup \mathbf{O}_{\mathbf{A}}^{t-1}) = \sum_{t'=t}^{t+l} \sum_{o \in O_{\mathbf{A}}^{t'}} \sum_{i=1}^{k} p_a \left( s_t^{o(l)} = i \mid \mathbf{O}_{\mathbf{A}}^{t'-1} \right)$$

The inner sum of this expression computes the expected loss prevented by making a single observation $o$. Thus, the utility received by the agents is equal to the expected loss they prevent, when the loss of a successful attack is assumed to be equal for each vertex of the layout graph. However, function $f$ can be made to reflect the non-homogeneous nature of the by multiplying the inner sum by the loss incurred from an attack on vertex $o(l)$.

#### 6.3.3.2 Experimental Setup

The experimental setup for this domain is identical to that of the pursuit-evasion domain in Section 6.3.2, including the algorithms used to benchmark our algorithm, with the following exceptions:

- Each problem instance lasts for 200 time steps; it does not terminate after an attacker is captured.
- New attacks appear with $p = 3 \cdot 10^{-4}$, and last $k = 20$ time steps.[15] These values were chosen to create problem instances that distinguish between poor performing and well performing algorithms; for example, by setting the attack probability too low, all algorithms will do fairly well, while by setting it too high, all algorithms will exhibit roughly equal poor performance.
- We measured the total loss incurred by successful attacks, instead of the capture time.

#### 6.3.3.3 Results

The results are shown in Figure 6.15 and Table 6.4 (which are analogous to Figure 6.14 and Table 6.3 for pursuit evasion). First of all, a comparison between Figures 6.15 and 6.14 shows that the algorithms now perform more uniformly. This is due to the fact that, in contrast to the pursuit evasion domain where there is no bound on the capture time, the patrolling domain has a bounded worst case performance, which occurs if none of the (finite) number of attackers is captured.

Again, we conclude that MS K-M outperforms TSP, Greedy, MS-8 and GGreedy in terms of median loss, and has a smaller statistical dispersion (i.e. a smaller inter-quartile

[15] Since the office environment has 350 vertices, the expected number of attacks is approximately 20.

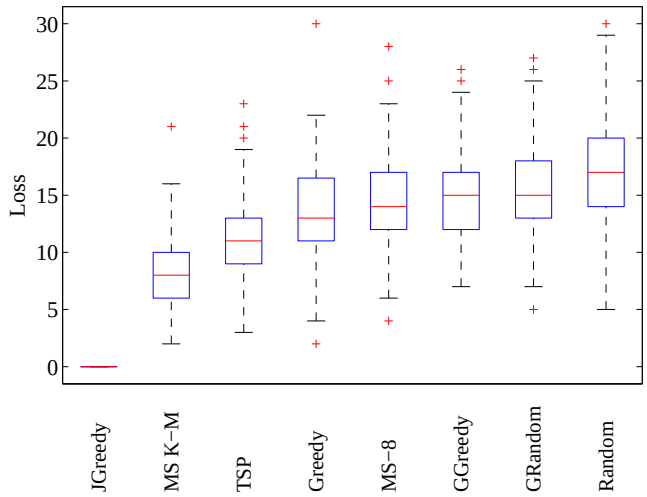



FIGURE 6.15: Empirical performance on 200 instances of the patrolling domain.

| | % Improvement (95% CI) | | | |
|---|---|---|---|---|
| **Environment** | **Greedy** | **GGreedy** | **MS-8** | **TSP** |
| Office | 43.7% | 46.9% | 45.2% | 30.0% |
| Lattice | 32.4% | 37.2% | 21.4% | 26.1% |

TABLE 6.4: Lower bounds of the 95% confidence intervals of performance increase of MS K-M compared to the four most competitive benchmarks in the patrolling domain.

range). Moreover, from Table 6.4, we conclude that the performance increase of MS K-M compared to these algorithms is statistically significant, and does not drop below 30% in the office environment.

This concludes the empirical evaluation of our algorithm in three different information gathering domains. We demonstrated the effectiveness of the techniques for ensuring network connectivity developed in Section 6.2.5, compared the two action selection heuristics proposed in Section 6.2.2, and concluded that heuristic 2, which uses graph clustering, is superior to heuristic 1, which considers paths in 8 fixed directions. Most importantly, we showed that our algorithm outperforms its competitors in all three domains.

## 6.4 Summary

In this chapter, we developed a decentralised coordination algorithm for receding horizon control of mobile agents. To achieve this, it uses the max-sum algorithm to periodically maximise the observation value received in the next $l$ time steps. We improved the scalability of the max-sum algorithm by exploiting the locality property of the observation value function to make the factor graph sparser, by carefully selecting the paths over

which agents coordinate to make the size of their action spaces more manageable, and by using the two generic pruning techniques we developed in this chapter to alleviate the main bottleneck of the max-sum algorithm.

Furthermore, since the algorithm operates on the generic problem definition in Chapter 3, it is applicable to a wide range of information gathering domains. We demonstrated this by extensively analysing its performance in three distinct domains, and concluded that it outperforms the state of the art, as well the greedy un-negotiated coordination for monitoring environmental phenomena, pursuit-evasion and patrolling.

More specifically, in terms of the design requirements stated at the start of this thesis, we can conclude the following:

**Quality:** The empirical results from Section 6.3 show that the algorithm outperforms its competitors by 50% in terms of RMSE for monitoring environmental phenomena, 30% in terms of capture time in the pursuit evasion domain, and 30% in terms of loss from successful intrusions in the patrolling domain.

**Adaptiveness:** Since the algorithm established receding horizon control, and continuously revises the computed paths, the agents are capable of responding to events that occur during their deployment. The team is therefore able to adapt to its environment and *a priori* unknown events.

**Robustness:** The algorithm is based on the max-sum algorithm, which is robust to message loss and failing agents.

**Autonomy:** The use of the max-sum algorithm maintains the *autonomy* of the agents.

**Scalability:** As discussed in Section 2.6.1, the computational demand of the max-sum algorithm scales favourably (i.e. with the number of neighbours, not with the size of the team). Moreover, to further improve its scalability, we developed two generic pruning techniques, reduced the number of dependencies between agents, and used heuristics to reduce the number of paths over which the agents negotiate.

**Modularity:** Although not specifically mentioned in this chapter, it is possible to define the action space of agents in such a way as to reflect their different capabilities, and to have different layout graphs $G$ for each type of agent. For example, the layout graph for a UGV is more sparse than that of a UAV to reflect that its motion is constrained by the terrain. By embedding both graphs in the same environment, and having the agents coordinate over paths in their respective graphs, the algorithm can be directly applied without further modifications.

**Performance Guarantees:** The algorithm developed in this chapter does not give performance guarantees.

Despite its robustness and adaptiveness, the lack of guarantees offered by this algorithm on its long term performance can limit its applicability in safety critical applications (such as disaster response, surveillance, etc.), because the existence of pathological behaviour can not be ruled out. Given this, in the next chapter, we will investigate how to address this shortcoming.

# Chapter 7

# Non-Myopic Control of Mobile Agents with Performance Guarantees

In the previous chapter, we developed an accurate, robust, and adaptive algorithm for decentralised control of mobile information gathering agents. This algorithm established receding horizon control and thus only considers solution quality over a finite number of time steps. Consequently, it can not give guarantees on long term solution quality. As mentioned before, the lack of guarantees, particularly on the worst-case behaviour, can be an obstacle for the application of this algorithm in safety critical and sensitive domains.

Hence, we identify a need for a non-myopic (i.e. infinite look-ahead) algorithm with theoretical guarantees on solution quality. Recent work has already addressed this need for single (Meliou et al. 2007) and multiple mobile agents (Singh et al. 2009) for environments that are static over time, or are changing at a rate that is negligible compared to the time required to traverse them. As a result, they fall short of dealing with the (possibly) rapid rate of change within the agents' environment that we consider here.

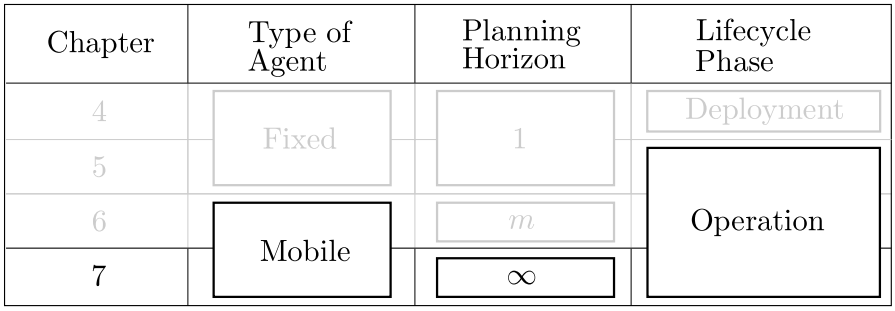



TABLE 7.1: The contributions of Chapter 7 in the context of the roadmap of this thesis.

More specifically, these algorithms do not consider the *temporality* property of observation value function $f$ (see Chapter 3). As a consequence, the algorithm proposed by Singh et al. (2009) computes finitely long paths, which tend not to return to previously visited locations, since no additional information (or value) can be obtained from doing so. In contrast, in continuously changing environments (which do exhibit temporality), it is imperative that agents periodically return to the same location so as to provide up-to-date situational awareness.

Against this background, in this chapter,[1] we develop an algorithm that computes near-optimal *patrols*: infinitely long paths designed to *continuously* monitor the environment. This algorithm is inspired by the divide and conquer algorithm proposed by Singh et al. (2009), which proceeds in three steps: it decomposes the environment into clusters, computes valuable paths within each of these clusters, and concatenates these paths to form a solution (see Section 2.5.2 for more details). Our algorithm follows a similar three step computation, but makes non-trivial modifications to operate in the aforementioned dynamic environments. The majority of these modifications are made in the third step. In particular, we utilise techniques from sequential decision-making to compute a policy that, given previously made observations, specifies which observations to make next. The execution of this policy yields the desired infinitely long patrol for a single agent.

We then use this algorithm to compute solutions for the multi-agent problem based on the method of *sequential allocation* proposed by Singh et al. (2009). This method computes a near-optimal joint policy for a set of agents, by greedily computing a policy for agent $\mathcal{A}_i$ conditioned on the previously computed policies for agents $\mathcal{A}_1, \ldots, \mathcal{A}_{i-1}$. This results in greatly reduced computational overhead compared to searching the joint policy space for $i$ agents. However, due to the different assumptions on which our work is based (i.e. continuously changing environments), the modifications we need to make to the single-agent algorithm are more involved than those described by Singh et al. (2009).

Finally, we explore the use of online decentralised coordination in an attempt to improve the agents' near-optimal joint policies, and provide an extensive empirical evaluation of the algorithm in two challenging information gathering scenarios.

In summary, the primary contributions of this chapter are as follows:[2]

- A non-myopic algorithm for computing near-optimal patrols for a single agent. The novelty of this algorithm lies in the fact that it computes infinite length paths for patrolling continuously changing environments (i.e. those that exhibit *temporality*).

[1] A journal paper about the research described in this chapter is in preparation and will be published as Stranders, Munoz de Cote, Rogers & Jennings (2010).

[2] Table 7.1 shows the context of these contributions in terms of the roadmap of this thesis.

- An algorithm for computing near-optimal patrols for multiple agents by greedily computing single-agent policies.
- Strong theoretical guarantees on both solution quality and computation cost of both the single-agent and multi-agent algorithms.
- An investigation into the use of online decentralised coordination algorithm to improve the multi-agent policy.
- Empirical analysis of the algorithm by benchmarking it against the receding horizon control algorithm developed in the previous chapter that does not give performance guarantees. We demonstrate that the non-myopic algorithm performs comparably in terms of average-case performance, and $> 10\%$ better in terms of worst-case performance. Moreover, we show that decentralised coordination results in a 5% increase in solution quality, but increases the number of searched states by approximately two orders of magnitude. We consider this evidence for the near-optimality of the multi-agent policy.

The remainder of the chapter is organised as follows. In Section 7.1 we define the problem of non-myopic information gathering by extending the problem formulation in Chapter 3. In section 7.2 we describe our algorithm for single and multiple information gathering agents. In Section 7.3 we derive bounds on the solution quality and the computational complexity of this algorithm. In Section 7.4 we propose two methods for improving the multi-agent policy through decentralised coordination. In Section 7.5 we empirically evaluate the algorithm and the decentralised coordination algorithms. Finally, in Section 7.6, we summarise the contributions made in this chapter, and assess them in terms of the design requirements of this thesis.

## 7.1 Problem Definition

We now formally describe the problem that we address in this chapter, by extending the problem formulated in Chapter 3 to non-myopic patrolling. Specifically, we define the solution to the observation maximisation problem in Equation 3.2 as a *policy*. In more detail, a policy $\pi$ specifies which observations $O^t_{\mathbf{A}}$ should be made in time step $t$, given observations $\mathbf{O}^{t-1}_{\mathbf{A}}$ that were taken in time steps before $t$ (subject to movement constraints imposed by layout graph $G$), i.e. $O^t_{\mathbf{A}} = \pi(\mathbf{O}^{t-1}_{\mathbf{A}})$.

To characterise the optimal policy $\pi^*$, we define a function $V_{\pi^*}(\mathbf{O}^t_{\mathbf{A}})$, which assigns a value to the state in which the agent has made observations $\mathbf{O}^t_{\mathbf{A}}$ under policy $\pi^*$. This value is equal to the discounted incremental value of observations $\pi^*$ makes in the future, given that observations $\mathbf{O}^t_{\mathbf{A}}$ have already been made:

$$V_{\pi^*}(\mathbf{O}_\mathbf{A}^t) = \rho_{\pi^*(\mathbf{O}_\mathbf{A}^t)}(\mathbf{O}_\mathbf{A}^t) + \gamma V_{\pi^*}\left(\mathbf{O}_\mathbf{A}^t \cup \pi^*(\mathbf{O}_\mathbf{A}^t)\right) \tag{7.1}$$

Since $\pi^*$ is the optimal policy, we know it selects the observations that maximise the discounted observation value. Thus, $\pi^*$ is fully defined by combining Equation 7.1 with:

$$\pi^*(\mathbf{O}_\mathbf{A}^t) = \arg\max_{O_\mathbf{A}^{t+1}} \left[\rho_{O_\mathbf{A}^{t+1}}(\mathbf{O}_\mathbf{A}^t) + \gamma V_{\pi^*}\left(\mathbf{O}_\mathbf{A}^t \cup \pi^*(\mathbf{O}_\mathbf{A}^t)\right)\right] \tag{7.2}$$

Note that policy $\pi^*$ is defined on the set of all possible observation histories $\mathbf{O}_\mathbf{A}^t$. The size of this set is exponential in the number of time steps $t$ that has elapsed, the average degree of layout graph $G$, and the number of agents $M$. As a result, computing the optimal policy is computationally intractable for all but the smallest of problems. Therefore, in this chapter we present an efficient algorithm that, instead of the optimal solution, computes a *near-optimal* policy, i.e. a policy that is guaranteed to be within a small factor of the optimal one.

## 7.2 Near-Optimal Non-Myopic Patrolling

In this section, we first develop a non-myopic algorithm for the single agent case (Section 7.2.1), which is later used as a building block for computing multi-agent policies (Section 7.2.2).

### 7.2.1 The Single-Agent Algorithm

The prime objective of the single-agent algorithm is to compress the exponentially large set of possible observation histories $\mathbf{O}_\mathbf{A}^t$ in Equation 7.2 into a more manageable sized set of world *states*, such that it becomes computationally feasible to conduct searches over the policy space. This is achieved as follows:

1. We exploit the *locality* property of an observation value function $f$ (see Chapter 3), by partitioning graph $G$ into clusters such that observations taken in different clusters are independent. The problem of maximising observation value can then be solved independently for each cluster.

2. We exploit the *temporality* property of $f$ by discarding observations older than $\tau$. These observations are independent of observations taken now or in the future, and can thus safely be ignored.

3. We divide time into intervals of length $B \in \mathbb{N}$, which is the number of time steps allocated to an agent to patrol a cluster. This parameter $B$ is chosen such that the agent can collect a *reasonable* amount of observation value.

By doing so, we can now represent the state of the world more compactly by keeping track of the number of time steps $\lambda_C \in \mathbb{N}$ that have elapsed since each cluster $C$ was patrolled by an agent. Since time is discretised into intervals of length $B$, we furthermore know that this number is necessarily a multiple of $B$, and that, whenever it exceeds $\tau$, any observation previously made within cluster $C$ has become stale and can be ignored.

The single-agent algorithm exploits each of these properties, and proceeds in three steps:

1. It partitions layout graph $G$ into a set of clusters $\mathbf{C} = \{C_1, \ldots, C_{|\mathbf{C}|}\}$ such that the distance between them is sufficient to ensure observations taken in different clusters are independent. Graph $G$ and clusters $\mathbf{C}$ are transformed into a bipartite graph $G[\mathcal{C}]$, which encodes the topological relations between the clusters. This graph defines the high-level constraints imposed on the movement of an agent between the clusters. For example, Figure 7.1 shows the clusters that can be identified in the layout graph of the IAM lab. In turn, Figure 7.2 shows $G[\mathcal{C}]$.

2. It computes *subpatrols* within each cluster. A subpatrol is a path within a single cluster of length $B$ along which a large amount of observation value is received. Each subpatrol corresponds to a movement allowed within $G[\mathcal{C}]$. For example, graph $G[\mathcal{C}]$ in Figure 7.2 allows the agent to move from $T_7$ through $C_6$ to $T_6$; a subpatrol for this movement is shown in 7.3.

3. It computes an optimal sequence in which clusters are visited by finding an optimal concatenation of subpatrols. In order to do this, we construct an MDP in which states represent the position of the agent, as well as the last visitation time $\lambda_C$ of each cluster $C$. A solution to this MDP is a policy that instructs the agent which cluster to patrol next, given its current position and the last time the clusters were visited. The concatenation of the corresponding subpatrols yields the desired patrol.

A more detailed description of each of these steps follows.

#### 7.2.1.1 Step 1: Partition the Layout Graph

The objective of the first step is to partition graph $G$ into a set of clusters, ensuring that the diameter[3] $D$ of each cluster is small enough such that any patrol of length $B$

[3]The diameter of a graph is the maximum shortest distance between any pair of its vertices.

**Algorithm 10** Algorithm for transforming layout graph $G$ into bipartite cluster graph $G_{\mathcal{C}}$.

**Require:** Layout graph $G = (V, E)$
**Require:** Maximum diameter $D$
**Ensure:** Cluster graph $G_{\mathcal{C}} = ((\mathbf{C} \cup \mathbf{T}), E_{\mathcal{C}})$, such that:

- $\mathbf{C} = \{C_1, \ldots, C_{|\mathbf{C}|}$ is a set of clusters;
- $\mathbf{T} = \{T_1, \ldots, T_{|\mathbf{T}|}$ is a set of transfer nodes;
- $\forall v \in C_i, \forall v' \in C_j, i \neq j : \mathrm{d}(v, v') \geq \delta$;
- $\forall C \in \mathbf{C} : \mathrm{diam}(C) \leq D$.

*Cluster graph G:*
1: $\mathbf{C} = Cluster(G, D)$
*Identify transfer nodes* $\mathbf{T}$*:*
2: $V_B = \bigcup_{i=1}^{|\mathbf{C}|} \{v \in C_i \mid \exists v' \in V : (v, v') \in E\}$
3: $\mathbf{T} = ConnectedComponents(G[V_B])$
4: $E_{\mathcal{C}} = \{(C, T) \mid C \in \mathbf{C}, T \in \mathbf{T}, \exists v \in C, \exists v' \in T : (v, v') \in E\}$
*Strip away vertices less than* $\frac{1}{2}\delta$ *away from vertices in different clusters:*
5: $V_\delta \leftarrow \varnothing$
6: **for** $C \in \mathbf{C}$ **do**
7: $\quad V_\delta \leftarrow V_\delta \cup \{v \in C \mid \exists v' \in V \setminus C : \mathrm{d}(v, v') \leq \frac{1}{2}\delta\}$
8: **end for**
9: **for** $C \in \mathbf{C}$ **do**
10: $\quad C \leftarrow C \setminus V_\delta$
11: **end for**
12: **return** $G_{\mathcal{C}} = ((\mathbf{C} \cup \mathbf{T}), E_{\mathcal{C}})$

visits at least $k$ vertices (reasons for these constraints will become clear in Section 7.3). The resulting partitions are transformed into a bipartite graph $G[\mathcal{C}]$ that encodes their topological relations.

Algorithm 10 shows the necessary steps. First, it partitions the graph into a set of clusters $\mathbf{C} = \{C_1, \ldots, C_{|\mathbf{C}|}\}$ (line 1). There are many different ways of doing this, however, we are interested in generating clusters with an upper bound on each intra-cluster diameter. With this in mind, we use the algorithm proposed by Edachery et al. (1999) as a subroutine,[4] which we will refer to as $Cluster(G, D)$. This algorithm takes as input a graph $G$ and a diameter $D$ and returns a set of clusters $\mathbf{C}$, such that for each cluster $C$, $\mathrm{diam}(C) \leq D$. In more detail, it solves a slight variation of the *pairwise clustering problem*. The pairwise clustering problem involves finding the smallest $k$-clustering of $G$, such that each of the $k$ clusters has a diameter smaller than $D$.

In the second step, the algorithm identifies the *transfer nodes* $\mathbf{T}$ of the clustering (lines 2–4). These transfer nodes are connected components of graph $G$ that are on the boundary of the clusters. To compute these, the algorithm identifies the boundary vertices $V_B$ of the clusters, i.e. those vertices that have at least one adjacent vertex in a different cluster.

[4] We choose this algorithm for its computational efficiency (see Section 7.3.2). However, depending on the type of graph, there might be other algorithms worth investigating. For an overview, we refer the reader to Schaeffer (2007).

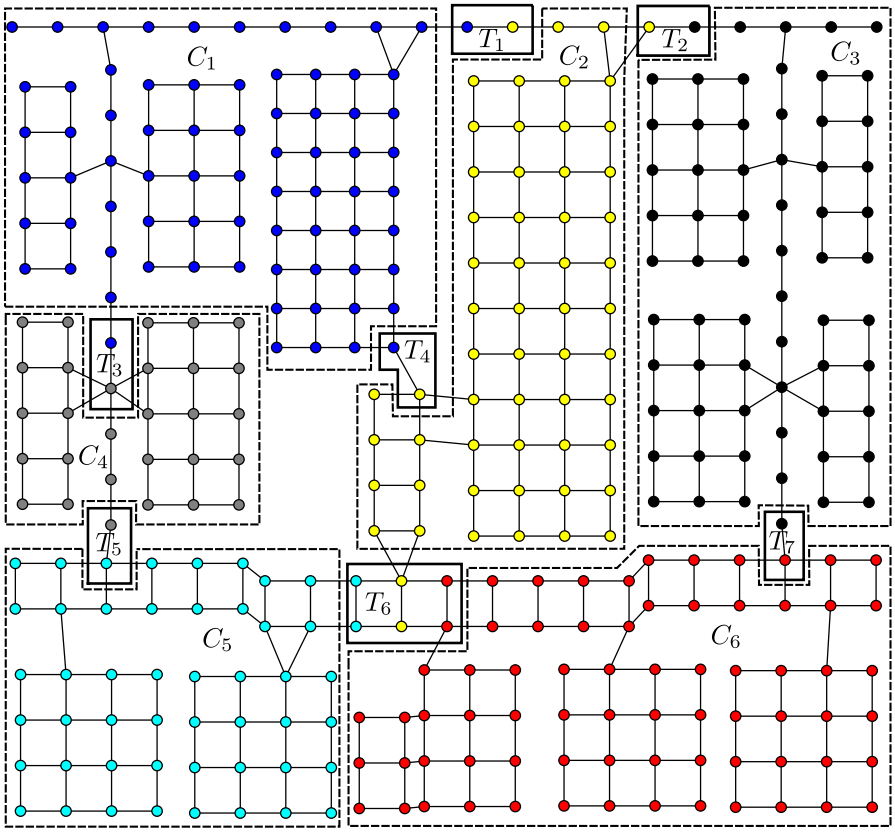



FIGURE 7.1: The clusters and transit nodes identified in the layout graph of the IAM lab. The dotted lines indicate the boundaries of the clusters $\mathbf{C}$ and the solid lines outline the seven transfer nodes $\mathbf{T}$ that connect the clusters.

Let $G[V_B]$ be the subgraph induced by $V_B$. The set of transfer nodes $\mathbf{T} = \{T_1, \ldots, T_{|\mathbf{T}|}\}$ corresponds to the set of connected components in $G[V_B]$.

The third step of the algorithm ensures independence of observations made in different clusters, by stripping away all vertices that are less than $1/2\delta$ away from vertices in other clusters (lines 5–11).

The resulting clusters, transfer nodes and their connections are represented as a bipartite graph $G[\mathcal{C}] = ((\mathbf{C} \cup \mathbf{T}), E_{\mathcal{C}})$, where an edge exists between a cluster $C \in \mathbf{C}$ and a transfer node $T \in \mathbf{T}$ if and only if the original graph $G$ contains an edge $e$ that has an endpoint in both $C$ and $T$ (line 4). This graph represents valid high-level movements between clusters, and is used in step 3 to define valid transitions within the MDP.

The following example illustrates the operation of this algorithm:

**Example 7.1.** *Figure 7.1 shows the six clusters and seven transfer nodes identified in the layout graph $G$ of the IAM lab. Figure 7.2 depicts the bipartite graph $G_{\mathcal{C}}$ which represents the interconnections between the clusters and transfer nodes in Figure 7.1.*

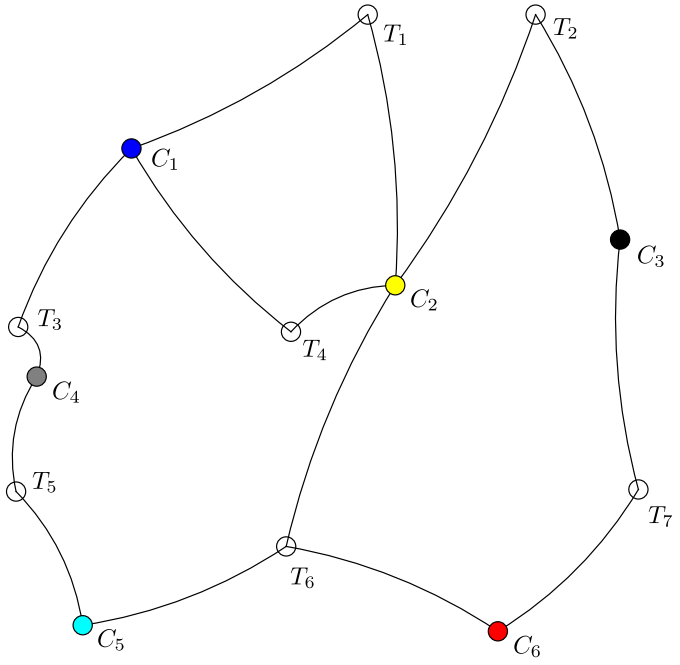



FIGURE 7.2: The bipartite graph $G[\mathcal{C}]$ that represents the topological relations between the clusters and transfer nodes in Figure 7.1

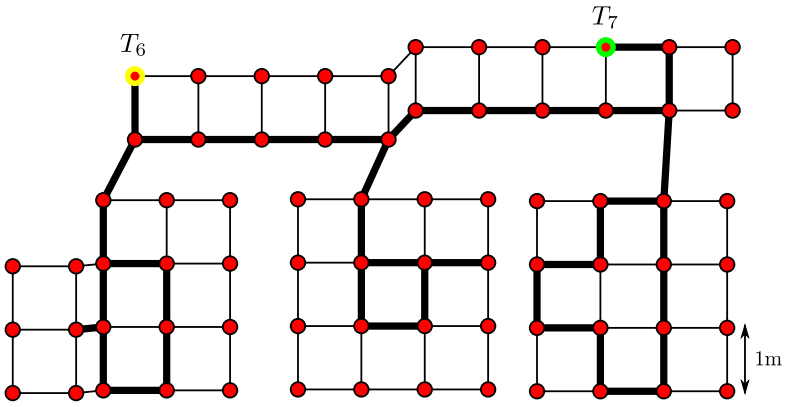



FIGURE 7.3: A patrol within cluster $C_6$ from transit node $T_7$ to $T_6$.

#### 7.2.1.2 Step 2: Compute Patrols

By clustering the layout graph, we have now decomposed the problem of finding a path of high value through the original (large) graph to a set of easier independent subproblems that involve finding paths within the (smaller) clusters. The second step of the algorithm uses a subroutine for computing these paths, which we will refer to as *subpatrols*. For each cluster, the subroutine is used to compute subpatrols between each pair of the cluster's adjacent transfer nodes. The reason for doing this is that in step three, agents will be allowed to enter from and exit to any of the adjacent transfer nodes of the cluster.

In more detail, the problem is now to find a sequence of vertices that maximises the

value of observations, subject to a finite budget $B$. Since this problem is NP-hard,[5] solving this problem optimally is computationally intractable for an arbitrarily large cluster. Therefore, instead, the patrolling subroutine is chosen to be approximate. That is, it computes subpatrols of near-optimal value with that are shorter than $B$.

In more detail, for a given cluster $C$, entry $T$ and exit $T'$, the subroutine (see Algorithm 11) proceeds in two steps. In the first step (lines 3–6), it orders the vertices of $C$ by their incremental value—the value obtained by greedily adding the observations $O_v$ made at $v$ to the already selected set $\mathbf{O}$, such that the incremental value $f(\mathbf{O} \cup O_v) - f(\mathbf{O})$ of observations collected at $v$ is maximised. This results in a greedy sequence $s_G = (v^{(1)}, \ldots, v^{(|C|)})$. In the second step (lines 7–15), it seeks to find a subpatrol $P_{T,C,T'}$ from $T$ to $T'$ with a length of at most $B$ and maximises the length $n$ of the prefix of $s_G$ (i.e. its first $n$ elements) that is visited along the path. This problem can be encoded as an instance of the Travelling Salesman Problem (TSP) where we seek to find a minimum cost cycle $(T, v^{(1)}, \ldots, v^{(n)}, T', T)$. Here, the cost of moving between two vertices $v_i$ and $v_j$ equals the length of the shortest path between them, and the cost of moving between $T$ and $T'$ equals 0. Since solving the Travelling Salesman Problem (TSP) itself is NP-hard, we use the heuristic algorithm by Christofides (1976), which has the best known performance guarantee ($\frac{3}{2}$) of any approximate algorithm for the TSP (Gross & Yellen 1999).

**Example 7.2.** *Consider an agent that is capable of perfectly observing all vertices within a sensing radius of* $1.5m$*. Value function $f$ is defined in terms of the number of vertices that are observed. Figure 7.3 shows the subpatrol $P_{T_7,C_6,T_6}$ computed by Algorithm 11 in the graph shown in 7.1 with $B = 50$. Note that this patrol is not optimal, in the sense that the same number of vertices could have been observed with a cost of* 44 *instead of* 46.

#### 7.2.1.3 Step 3: Compute the Patrolling Strategy

In the third and final step of the algorithm, we define and solve a MDP over the simplified patrolling problem in graph $G_{\mathcal{C}}$. By so doing, we exploit the clustering of the layout graph and the subpatrols within each of the clusters of the graph. The result of the final step is a policy for a single agent.

This (deterministic) MDP is a 4-tuple $(S, A, \delta(\cdot,\cdot), R(\cdot,\cdot))$ where:

- $S$ is a set of states encoding the current position of the agent and the time each cluster was last visited.
- $A$ is a set of actions. In this context, each action in this set corresponds to a subpatrol that start from the agent's current position.

[5] It is easy to see that the Travelling Salesman Problem can be reduced to this problem.

**Algorithm 11** Computing a subpatrol of cluster $C$ from entry $T$ to exit $T'$.

**Require:** Cluster $C$, transit node $T$ (entry), transit node $T'$ and budget $B$
**Ensure:** A patrol $P_{T,C,T'}$ of $C$ from $T$ to $T'$ with cost $c(P_{T,C,T'}) < B$
*Sort vertices on their greedy observation value:*
$s_G \leftarrow ()$
**while** $C \setminus s_G \neq \emptyset$ **do**
  Let $O_v$ be the observations made at $v$, and $\mathbf{O} = \bigcup_{v \in s_G} O_v$.
  $s_G \leftarrow s_G || \arg\max_{v \in C \setminus s_G} f(\mathbf{O} \cup O_v) - f(\mathbf{O})$
**end while**
*Find the maximum $n$ such that the cost of the subpatrol that visits the first $n$ elements of $s_G$ does not exceed $B$:*
$n \leftarrow 0$
$P' \leftarrow (T, T')$
**repeat**
  $n \leftarrow n + 1$
  $P_{T,C,T'} \leftarrow P'$
  $s_G^n \leftarrow \text{prefix}(s_G, n)$ {Select first $n$ elements of $s_G$}
  $P' \leftarrow TSP(T, s_G^n, T')$
**until** $c(P') > B$
**return** $P_{T,C,T'}$

- $s' = \delta(s, a)$ is the state obtained from performing action (subpatrol) $a$ in state $s$. Thus, $\delta$ is a deterministic transition function.
- $R(s, a)$ is the observation value received by performing action (subpatrol) $a$ in state $s$.

In what follows, we discuss each element in more detail.

**State Space** The state space $S$ consists of states of the form $(T, \boldsymbol{\lambda})$ that record the current position of the agent as well as the number of elapsed time steps since each cluster was last patrolled. The first element, the agents' position, is one of the transfer nodes $T \in \mathbf{T}$, because the agent exits to an adjacent transfer node, each time it patrols a cluster. The second element is a vector $\boldsymbol{\lambda} = [\lambda_{C_1}, \ldots, \lambda_{C_{|\mathbf{C}|}}]$, each element of which never exceeds $\tau$ time steps, since observations made longer than $\tau$ time steps ago are independent of new observations made within that cluster (through the *temporality* property). Now, keeping track of the exact number of time steps since a cluster was last visited yields $\tau^{|\mathbf{C}|}$ distinct possible states, causing the problem to become intractable for even a very small number of clusters or $\tau$. However, by exploiting the knowledge that an agent takes $B$ time steps to patrol a cluster, and if we furthermore choose $B$ as divisor of $\tau$, we can ensure that $\lambda_C \in \{0, \frac{\tau}{B}, \frac{2\tau}{B}, \ldots, \tau\}$. This drastically reduces the number of distinct possible visitation states of a single cluster from $\tau + 1$ to $\frac{\tau}{B} + 1$. Combining this with the possible positions of the agent, the state space contains at most $|\mathbf{T}|(\frac{\tau}{B} + 1)^{|\mathbf{C}|}$ states.

**Action Space** The action space $A(s)$ in state $s$ consists of all subpatrols that can start from the agent's current position. Thus, in state $s = (T, \boldsymbol{\lambda})$, $A(s)$ contains all sequences $(T, C, T')$ (corresponding to subpatrol $P_{T,C,T'}$) where $C$ is an adjacent cluster of transfer node $T$, and, in turn, $T'$ is an adjacent transfer node of cluster $C$.

**Example 7.3.** *The valid actions in the graph shown in Figure 7.1 for state* $s = (T_1, \cdot)$ *are* $A(s) = \{(T_1, C_1, T_1), (T_1, C_1, T_3), (T_1, C_2, T_1), (T_1, C_2, T_2), (T_1, C_2, T_4), (T_1, C_2, T_6)\}$.

**Reward Function** The reward received for performing action $(T, C, T')$ in state $s = (T, [\lambda_{C_1}, \ldots, \lambda_{C_{|\mathbf{C}|}}])$ is the value of the observations made along subpatrol $P_{T,C,T'}$, given that cluster $C$ was visited $\lambda_C$ time steps ago. Since it is unknown which subpatrol was previously used to visit $C$, we assume that all vertices of $C$ were visited simultaneously $\lambda_C$ time steps ago, at which point a set of observations $\mathbf{O}_{\mathbf{C}}^{-\lambda_C}$ was made. Thus, the incremental value of the observations made along $P_{T,C,T'}$ with respect to $\mathbf{O}_{\mathbf{C}}^{-\lambda_C}$ yields a conservative estimate (i.e. lower bound) on the true reward for action $(T, C, T')$.

More formally, the reward $R(s, a)$ of performing $a = (T, C, T')$ in state $s = (T, [\lambda_{C_1}, \ldots, \lambda_{C_{|\mathbf{C}|}}])$ is the sum of incremental values of observations made along subpatrol $P_{T,C,T'} = (T, v^{(1)}, \ldots, v^{(n)}, T')$. Note that $R(s, a)$ depends solely on the visitation state $\lambda_{C_i}$ of cluster $C_i$ and the entry $T$ and exit $T'$; the visitation states of the clusters other than $C$ are irrelevant for computing the action's reward. Thus, we can define a function $I(C, \lambda_C, T, T')$ that computes the value of a subpatrol, such that:

$$\begin{aligned} R(s, a) &= I(C, \lambda_C, T, T') \\ I(C, \lambda_C, T, T') &= \sum_{i=1}^{n} \gamma^{t_i} \cdot \rho_{O_{v^{(i)}}} \left( \bigcup_{j=1}^{i-1} O_{v^{(j)}} \cup \mathbf{O}_C^{-\lambda_C} \right) \end{aligned} \tag{7.3}$$

Here, $\mathbf{O}_C^{-\lambda_C}$ denotes the set of observations made $\lambda_C$ time steps ago at each vertex of $C$, the set $O_v$ denotes the observations made at $v$ (as before), and $t_i$ is the time at which $v^{(i)}$ is visited, which is the time it takes to arrive at $v^{(i)}$ traversing subpatrol $P_{T,C,T'}$:[6]

$$t_i = \sum_{j=1}^{i-1} \mathrm{d}_G \left( v^{(j)}, v^{(j+1)} \right)$$

**Transition Function** The transition function formalises how the MDP transitions under a given action $a$. When an agent patrols a cluster $C_i$ by performing action $a = (T, C, T')$, the process transitions to state $s'$, in which the agent is positioned at $T'$ and the visitation time of the cluster $\lambda_C$ is reset to 0. Furthermore, since the agent takes $B$ time steps to patrol a cluster, the visitation times of clusters $C_j$ $(j \neq i)$ is incremented

[6] Recall from Chapter 3 that $\mathrm{d}_G(v, v')$ is the length of the shortest path in $G$ from $v$ to $v'$.

by $B$, if not already equal to $\tau$. Deterministic function $\delta(s, a) = s'$ formalises the transition from state $s = (T, [\lambda_{C_1}, \ldots, \lambda_{C_{|\mathbf{C}|}}])$ under action $a = (T, C_i, T')$:

$$\delta(s, a) = (T', [\hat{\lambda}_{C_1}, \ldots, \hat{\lambda}_{C_{i-1}}, 0, \hat{\lambda}_{C_{i+1}}, \ldots, \hat{\lambda}_{C_{|\mathbf{C}|}}])$$

where $\hat{\lambda}_{C_1} = \min(\lambda_{C_1} + \frac{\tau}{B}, \tau)$.

This transition function enables us to reduce the size of the state space defined earlier, by only considering the states that are reachable from a given initial state $\overline{s} = (T, [\tau, \ldots, \tau])$ in which none of the states have been visited yet. For example, an unreachable state in the setting of Figure 7.1 is $(T_1, [\tau, \tau, 0, \tau, \tau, \tau])$ that encodes that cluster $C_3$ was just patrolled by the agent which subsequently moved to a transit node that is inaccessible from $C_3$. We define the set of *reachable* states $S_r(s)$ from state $s$ as:

$$S_r(s) = \{s\} \cup \bigcup_{a \in A(s)} S_r(\delta(s, a)) \tag{7.4}$$

**Solving the MDP** A solution to the MDP $(S, A, \delta(\cdot, \cdot), R(\cdot, \cdot))$ defined above is a policy of the form $\pi(s) = a$ that, for every possible state $s \in S$, yields action $a$ that maximises the expected discounted reward. This policy is characterised by the following equations:

$$\pi(s) = \arg\max_a \{R(s, a) + \hat{\gamma} V(\delta(s, a))\} \tag{7.5}$$

$$V(s) = R(s, \pi(s)) + \hat{\gamma} V(\delta(s, a)) \tag{7.6}$$

Here, $V(s)$ is referred to as the *state value* of $s$ under policy $\pi$, which equals the discounted sum of rewards to be received by following policy $\pi$ from state $s$. Many algorithms can be used to compute policy $\pi$, such as policy iteration (Howard 1960), modified policy iteration (Puterman & Shin 1978), and prioritised sweeping (Moore & Atkeson 1993). However, one of the simplest is value iteration (Puterman 1994). This algorithm repeatedly applies following update rule:

$$V(s) = \max_a \{R(s, a) + \hat{\gamma} V(\delta(s, a))\} \tag{7.7}$$

until the maximum difference between two successive state value falls below a predefined threshold $\epsilon > 0$. After termination, the value of each state under policy $\pi$ is within $\epsilon$ of the optimal value.

### 7.2.2 The Multi-Agent Algorithm

We now show how to adapt the single-agent algorithm to compute policies for the multi-agent problem.

A straightforward way of doing this is to extend the MDP constructed in the single-agent algorithm to multiple agents. The state space of this multi-agent MDP contains the position of each agent, and its action space is defined as the Cartesian product of the action spaces of the single agents. However, in so doing, the size of the state and action space grow exponentially with the number of agents $M$, allowing only the smallest of problem instances to be solved. Instead, our approach computes a (nearly) optimal joint policy for a team of agents, by computing a single-agent policy for each $\mathcal{A}_i$, conditioned on the previously computed policies of agents $\mathbf{A}_{-i} = \{\mathcal{A}_1, \ldots, \mathcal{A}_{i-1}\}$.

This method is equivalent to *sequential allocation* of multiple agents proposed by Singh et al. (2007). However, the problem they address is to compute finitely long paths for each agent, instead of policies. Under this assumption, it is possible to define a new observation value function $f'$ that assigns value to observations $O_i$ made by agent $\mathcal{A}_i$ conditioned on observations made by agents $\mathbf{A}_{-i}$, i.e.:

$$f'(O_i) = \rho_{O_i}\left(\bigcup_{j=1}^{i-1} O_j\right)$$

However, this implicitly assumes there exists an order in which the agents make observations: agent $\mathcal{A}_1$ traverses the environment first, $\mathcal{A}_2$ second, etcetera. Unfortunately, no such ordering is possible with paths of *infinite* length (i.e. the policies computed by the single-agent algorithm). Thus, we need to make non-trivial modifications to the algorithm developed in the previous section in order to achieve the same effect.

To do this, we modify the MDP defined in Section 7.2.1.3, such that the goal of agent $\mathcal{A}_i$ becomes to collect the observation value left behind by agents $\mathbf{A}_{-i}$. To accomplish this, we make its transition function dependent on the policies of agents $\mathbf{A}_{-i}$. In turn, these agents are unaware of the existence of agent $\mathcal{A}_i$, meaning that their beliefs (i.e. state) about the world are unaffected by agent $\mathcal{A}_i$'s actions.

The new MDP is obtained from the single-agent MDP discussed in the previous section by making the following modifications:

**State Space** Agent $\mathcal{A}_i$ now takes into account the policies of agents $\mathbf{A}_{-i}$ (but not vice versa). Since these these policies operate on the states of these agents, we need to include the states of $\mathbf{A}_{-i}$ into the state of agent $\mathcal{A}_i$. States thus become composite (or recursive).

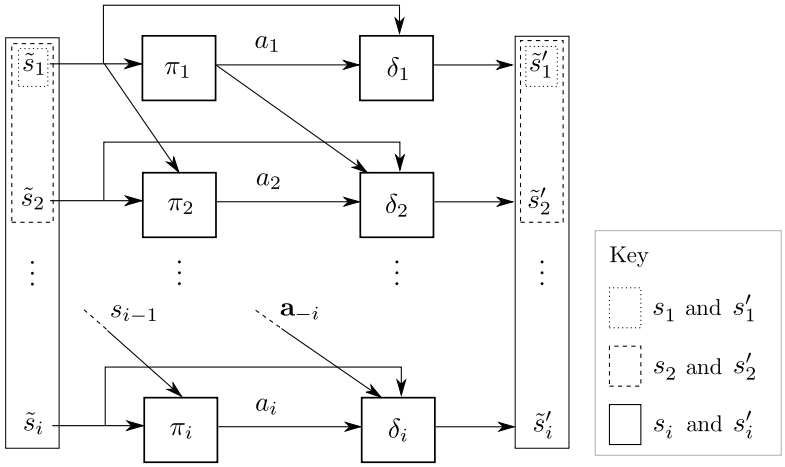



FIGURE 7.4: The recursive state space of agent $\mathcal{A}_i$.

**Transition Function** The transition function now reflects the effect of agent $\mathcal{A}_i$'s actions, as well as those of agents $\mathbf{A}_{-i}$.

**Reward Function** The reward function now rewards agent $\mathcal{A}_i$ only for the observation value left behind by agents $\mathbf{A}_{-i}$.

The relations between states, policies and transition functions in this modified MDP are shown in Figure 7.4. In the remainder of this section we shall discuss each modification in more detail.

**State Space** The new MDP takes into account the effect of agent $\mathcal{A}_i$'s actions, as well as those of agents $\mathbf{A}_{-i}$ who are executing their policies beyond agent $\mathcal{A}_i$'s control. In order to determine these actions, the MDP needs to include knowledge of the policies of agents $\mathbf{A}_{-i}$, as well as their current states.

Thus, we define composite states, which combines the *atomic* state—the states of the single-agent MDP defined in Section 7.2.1.3—of $\mathcal{A}_i$ with the composite state of $\mathcal{A}_j$. Let $\tilde{s}$ denote the atomic states of the form $(T, \boldsymbol{\lambda})$ as in Section 7.2.1.3. The state of agent $\mathcal{A}_i$ is given by the following recursive relation:

$$\begin{aligned} s_1 &= \quad \tilde{s}_1 \\ s_2 &= \quad (\tilde{s}_2, s_1) \\ &\vdots \\ s_i &= \quad (\tilde{s}_i, s_{i-1}) \end{aligned}$$

**Transition Function** To determine the successor state $s'_i$ obtained by applying action $a_i$ of agent $\mathcal{A}_i$, the transition function first determines the state $s'_{i-1}$ that results from the actions of agents $\mathbf{A}_{-i}$. State $s'_i$ is then obtained by applying action $a_i$ to $s'_{i-1}$.

With this in mind, we define the transition function $\delta_i$ for agent $\mathcal{A}_i$ as follows:

$$\begin{aligned} s'_1 &= \delta_1(s_1, a_1) \\ s'_2 &= \delta_2(s_2, \pi_1(s_1), a_2) \\ &\vdots \\ s'_i &= \delta_i(s_i, \pi_1(s_1), \pi_2(s_2), \ldots, a_i) \end{aligned}$$

The following example demonstrates the new state space and transition function.

**Example 7.4.** *Consider the environment in Figure 7.1 and bipartite graph $G[\mathcal{C}]$ in Figure 7.2 with two agents. At time step $t$, the atomic states $\tilde{s}$ of these agents are $\tilde{s}_1 = (T_7, [\tau, \tau, \tau, \tau, \tau, 0])$ and $\tilde{s}_2 = (T_6, [\tau, \tau, 0, \tau, \tau, 0])$ (and the composite state of $\mathcal{A}_2$ is $s_2 = (\tilde{s}_2, s_1)$). Thus, agent $\mathcal{A}_1$ has just patrolled cluster $C_6$ and is now positioned at $T_7$. Similarly, agent $\mathcal{A}_2$ has just patrolled cluster $C_3$ and is now positioned at $T_6$. Note that agent $\mathcal{A}_2$ is aware of the fact that agent $\mathcal{A}_1$ patrolled $C_6$, but agent $\mathcal{A}_1$—being unaware of the existence of agent $\mathcal{A}_2$—does not know about the new state of cluster $C_7$.*

**Reward Function** The modifications that need to be made to the reward function can best be explained by demonstrating what happens when we use the reward function from Section 7.2.1.3 in its current form. The effect of this is twofold. First, it results in *synchronous* double counting, which occurs when two agents patrol the same cluster within the same time step. In this case the reward for patrolling the cluster is received twice. The second effect, *asynchronous* double counting, is a little more subtle. For ease of exposition, we will illustrate this with an example.

**Example 7.5.** *(Continued from Example 7.4) At time step $t$, agent $\mathcal{A}_1$ patrols $C_3$ by choosing action $(T_7, C_3, T_2)$ and transitions to $(T_3, [\tau, \tau, 0, \tau, \tau, 0])$. The reward for this transition is equal to the observation value obtained from patrolling cluster $C_3$ in state $\tau$. In reality, however, much less value is obtained, since agent $\mathcal{A}_2$ patrolled $C_3$, and reset its visitation time $\lambda_3$ to 0. In a way, agent $\mathcal{A}_2$ "stole" agent $\mathcal{A}_1$'s reward for patrolling $C_3$.*

In general, asynchronous double counting occurs whenever an agent $\mathcal{A}_i$ patrols a cluster $C$ before agent $\mathcal{A}_j$ $(j < i)$, such that $\mathcal{A}_j$'s belief of $\lambda_C$ at the moment of patrolling $C$ is less than its true value.

To prevent double counting—both synchronous and asynchronous—we introduce a penalty $P$ for agent $\mathcal{A}_i$ that compensates for the reduction of reward of the agent $\mathcal{A}_j$ $(j < i)$ that patrols $C$ next, as follows:

$$R_i(s, (T, C, T')) = R(s, (T, C, T')) - P \tag{7.8}$$

| | Actual Reward | | Marginal Contribution | |
|---|---|---|---|---|
| **Time step** | $\mathcal{A}_1$ | $\mathcal{A}_2$ | $\mathcal{A}_1$ | $\mathcal{A}_2$ |
| $t_0$ | 1 | | 1 | |
| $t_3$ | | $0.4\gamma^3$ | | $0.4\gamma^3 - 0.6\gamma^6$ |
| $t_6$ | $0.4\gamma^6$ | | $\gamma^6$ | |
| **Total** | $1 + 0.4\gamma^3 + 0.4\gamma^6$ | | $1 + 0.4\gamma^3 + 0.4\gamma^6$ | |

TABLE 7.2: The actual and marginal rewards received by the agents in Example 7.6.

Here, $R(\cdot, \cdot)$ is the reward function defined in Section 7.2.1.3, and $P$ is the loss incurred by agent $\mathcal{A}_j$ ($j < i$) that will patrol cluster $C$ next. This is the (discounted) difference between the expected reward (which agent $\mathcal{A}_j$ would have received in the absence of agent $\mathcal{A}_i$) and its actual reward, discounted by the number of time steps $t_n$ that will elapse before agent $\mathcal{A}_j$ patrols $C$:

$$P = \gamma^{t_n}(R_{expected} - R_{actual}) \tag{7.9}$$

The rewards $R_{expected}$ and $R_{actual}$ are defined as:

$$R_{expected} = I(C, \min(\tau, \hat{\lambda}_C + t_n), \hat{T}_{start}, \hat{T}_{end}) \tag{7.10}$$

$$R_{actual} = I(C, t_n - B, \hat{T}_{start}, \hat{T}_{end}) \tag{7.11}$$

where $I(C, \lambda_C, T, T')$ is the value of a subpatrol (Equation 7.3), $\hat{\lambda}_C$ is the last visitation time of cluster $C$ in agent $\mathcal{A}_j$'s current state; $\hat{T}_{start}$ and $\hat{T}_{end}$ are the entry and exit transfer nodes chosen by agent $\mathcal{A}_j$ for its next visit to $C$.

The following example illustrates the behaviour of the new reward function.

**Example 7.6.** *Consider a scenario with two agents and a single cluster $C$. Agent $\mathcal{A}_1$ patrols this cluster at $t = 0$ and $t = 6$, and agent $\mathcal{A}_2$ at $t = 3$. Furthermore, suppose that the maximum reward for patrolling $C$ is 1, that $\tau = 6$ and that the reward increases 0.2 every time step the cluster is not patrolled. Figure 7.5 shows the function of* potential *reward as a function of time for this scenario, which is realised only when the cluster is patrolled. The two lines in Figure 7.5 represent the beliefs agents $\mathcal{A}_1$ and $\mathcal{A}_2$ have of this reward.*

*The rewards received by the agents are as follows (see Table 7.2). First, agent $\mathcal{A}_1$ patrols $C$ at $t = 0$ and receives a reward of* 1*. Second, agent $\mathcal{A}_2$ patrols the cluster at $t = 3$ and receives a reward of* 0.4*. At this point, the beliefs of the agents diverge, because agent $\mathcal{A}_1$ is not aware of agent $\mathcal{A}_2$'s actions. Finally, agent $\mathcal{A}_1$ patrols the cluster at $t = 6$. Contrary to its beliefs, it receives a reward of* 0.4 *instead of* 1*. In total the team receives a (discounted) reward of $1 + 0.4\gamma^3 + 0.4\gamma^6$.*

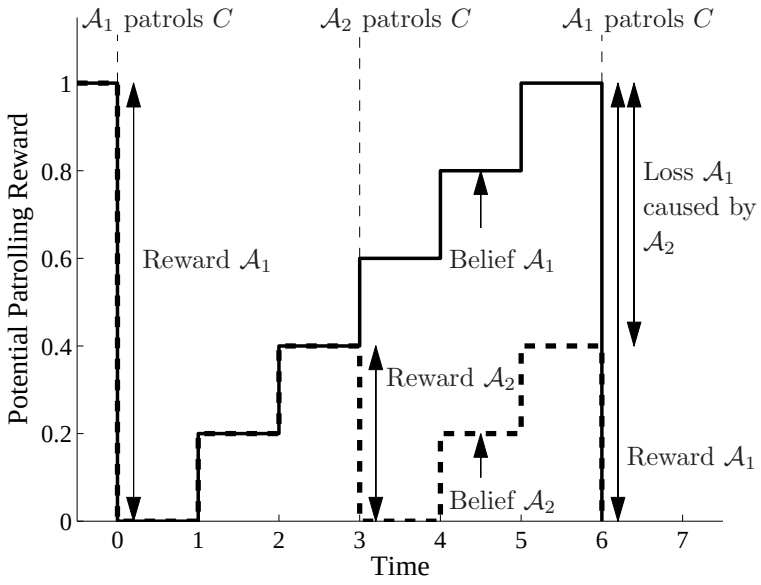



FIGURE 7.5: The potential reward for patrolling cluster $C$ in the scenario of Example 7.6.

*Now, consider the marginal contributions of the agents, i.e. the additional observation value received by adding an agent. To compute these for agent $\mathcal{A}_1$, we need only consider the beliefs of agent $\mathcal{A}_1$. It patrols the cluster twice when the reward equals 1, so its contribution is $1 + \gamma^6$. For agent $\mathcal{A}_2$, we need to consider its reward for patrolling the cluster at time step 3, but also the loss of reward of agent $\mathcal{A}_1$ at time step 6 it is responsible for. This loss is $0.6\gamma^6$, which makes its contribution $0.4\gamma^3 - 0.6\gamma^6$. Note that the sum of marginal contributions is equal to the sum of actually received rewards, as desired.*

This concludes the definition of the MDP for multiple agents. Using value iteration to solve this MDP (as before), we obtain a single policy for each agent. These policies are not optimal, since the policy for agent $\mathcal{A}_i$ is computed in a greedy fashion with respect to the policies of agents $\mathbf{A}_{-i}$. However, we can derive performance guarantees on their observation value, as we show in the next section.

## 7.3 Theoretical Analysis

In this section, we will derive bounds on both the solution quality of the algorithm presented in the previous section, as well as its computation overhead.

### 7.3.1 Solution Quality

We will first derive a lower bound on the solution quality of the single-agent algorithm, by proving the following lemma:

**Lemma 7.1.** *If* $\operatorname{diam}(C) \leq D = \frac{2}{3}B\left(\sqrt{\frac{\pi k}{2}} + \mathcal{O}(1)\right)^{-1}$*, Algorithm 11 computes a subpatrol* $P_{T,C,T'}$ *with an observation value* $I(C, \lambda_C, T, T')$ *of at least* $\gamma^B\left(1 - \left(\frac{k-1}{k}\right)^k\right) f(O^*)$*. Here,* $f(O^*)$ *is the value of the optimal set of observations made at* $k$ *vertices of* $C$*, ignoring the movement constraints of* $G$*.*

*Proof.* The proof consists of two steps. In the first step, we use a result by Moran (1984) to prove that any TSP in a graph with $k$ vertices with diameter $D$ has a cost less than $B$. Moran (1984) proved a bound on the length $L$ of the TSP of an arbitrary graph with $k$ vertices. Specifically, for a graph $G$ embedded in two-dimensional Euclidian space, the following relation holds:

$$L \leq \left(\sqrt{\frac{\pi k}{2}} + \mathcal{O}(1)\right) \operatorname{diam}(G)$$

By applying this relation to line 11 of Algorithm 11, we know that $n \geq k$ holds when this algorithm terminates. The extra cost of including $T$ and $T'$ (which are contained in $C$) into the TSP is compensated by the fact that we set the cost of moving between $T$ and $T'$ to 0 (since we require a *path* from $T$ to $T'$, not a cycle). As mentioned earlier, instead of solving the TSP optimally (which is an NP-hard problem), we use the approximation algorithm by Christofides (1976). This algorithm has an approximation ratio of $\frac{3}{2}$, which accounts for the factor of $\frac{2}{3}$ on bound of the $\operatorname{diam}(C)$.

In the second step of this proof, we apply Theorem 2.6 (see Section 2.5.1) proved by Nemhauser & Wolsey (1978) for obtaining a bound on the value of the greedily selected vertices (lines 2–5). This theorem states that the ratio between the value of the first $k$ greedily selected elements and the value of the optimal $k$ elements is at least $1 - \left(\frac{k-1}{k}\right)^k$. The factor of $\gamma^B$ stems from the fact that it is unknown in which order these $k$ elements are visited by the TSP. However, it is known that these elements are visited within $B$ time steps. Thus, we discount the incremental values obtained at these $k$ elements by $B$ time steps, which completes the proof. □

Step three of the algorithm (Section 7.2.1.3) uses these subpatrols and concatenates them into a single overarching patrol. The problem of finding an optimal sequence of subpatrols is represented as an MDP, which is optimally solved by value iteration. Consequently, the following holds for the value of the initial state $\overline{s}$, which is equal to the discounted observation value received by the agent by following policy $\pi$ (Equation 7.6):

$$V_\pi(\overline{s}) \geq \frac{\gamma^B}{1 - \gamma^B}\left(1 - \left(\frac{k-1}{k}\right)^k\right) f_{\min}(O^*) \tag{7.12}$$

where $f_{\min}(O^*)$ is the minimum value of $f_{\min}(O^*)$ over all clusters $\mathbf{C}$.

To prove a bound on the solution quality of the multi-agent algorithm, we prove that the observation value of a set of policies is submodular. To do this, we define a set function $g$ over a set of *single-agent* policies $[\pi_1, \ldots, \pi_M]$, that computes the discounted observation value of a set of policies:

$$g(\pi_1, \ldots, \pi_M) = \sum_{i=1}^{M} V_{\hat{\pi}_i}(\overline{s}_i)$$

Here, $\hat{\pi}_i$ is a policy for $\mathcal{A}_i$ of the form discussed in Section 7.2.2), which behaves identically in the presence of agents $\mathcal{A}_1, \ldots, \mathcal{A}_{i-1}$ as policy $\pi_i$ does in isolation. Thus, policy $\hat{\pi}_i$ visits the same clusters as $\pi_i$, and in the same order. Since the discounted marginal observation value of a single policy $\hat{\pi}_i$ received from initial state $\overline{s}$ is equal to $V_{\hat{\pi}_i}(\overline{s})$, function $g$ computes the discounted observation value of a team of agents $\mathcal{A}_1, \ldots, \mathcal{A}_M$.

We can now state the following result:

**Lemma 7.2.** *Function $g$ is a non-decreasing submodular set function.*

*Proof.* The non-decreasing property follows trivially from the fact that adding more agents never reduces the observation value they receive as a team. To prove submodularity, we need to show that, for every set of policies $\boldsymbol{\pi}' \subseteq \boldsymbol{\pi}$ and policy $\pi \notin \boldsymbol{\pi}'$ the following holds:

$$g(\{\pi\} \cup \boldsymbol{\pi}') - g(\boldsymbol{\pi}') \geq g(\{\pi\} \cup \boldsymbol{\pi}) - g(\boldsymbol{\pi})$$

To prove that this holds, we just need to prove that adding a policy $\pi$ to a set of policies $\boldsymbol{\pi}$ instead of $\boldsymbol{\pi}' \subseteq \boldsymbol{\pi}$ reduces reward and increases penalty (Equation 7.8). To prove the former, observe that $\mathcal{A}_i$'s belief of the last visitation time $\lambda_C^i$ of cluster $C$ is non-increasing in $i$, and Equation 7.3 is non-increasing in $\lambda_C^i$. Thus, adding predecessors to $\mathcal{A}_i$ reduces its reward for any subpatrol in any cluster. To prove the latter, observe that, with additional predecessors, the number of time steps $t_n$ before any predecessor visits the same cluster $C$ decreases or stays the same. Since penalty $P$ is a strictly increasing function of $t_n$ (see Equations 7.9, 7.10, and 7.11), adding $\pi$ to $\boldsymbol{\pi}$ instead of $\subseteq \boldsymbol{\pi}$ indeed increases the penalty. □

Since the multi-agent algorithm maximises the incremental value of $g$ by greedily computing a policy of agent $\mathcal{A}_i$ with respect to the policies of $\mathcal{A}_1, \ldots, \mathcal{A}_{i-1}$, Theorem 2.6 by Nemhauser & Wolsey (1978) can be directly applied to obtain the following result:

**Corollary 7.3.** *For $M$ agents, the policies computed by the multi-agent algorithm are at least $\left(1 - \left(\frac{M-1}{M}\right)^M\right)$ as valuable as the optimal $M$ policies.*

### 7.3.2 Computational Complexity

The computational complexity of the algorithm can be decomposed into the complexity of its three steps:

- The computational complexity of phase 1 is determined by the complexity of the subroutine $Cluster(G, D)$ (line 1 in Algorithm 10), which partitions $G$ into $|\mathbf{C}|$ clusters such that each of these has a diameter at of most $D$. Unfortunately, the problem of finding such a partitioning that minimises $|\mathbf{C}|$ is a known NP-hard problem (Schaeffer 2007). To ensure computational efficiency, we choose an approximation algorithm that requires more than the optimum number of clusters to satisfy the maximum diameter requirement. In particular, as mentioned in Section 7.2.1.1, we select the algorithm proposed Edachery et al. (1999), which finds a partitioning in time $O(|V|^3)$.

- The majority of the computation required in step 2 is attributable to computing the TSP in line 12 in Algorithm 11. As mentioned earlier, the complexity of computing an optimal TSP is exponential in $|V|$ (assuming P $\neq$ NP). However, if we use the heuristic proposed by Christofides (1976), which has the most competitive performance bounds, this is reduced to $O(|V|^3)$.

- Lastly, in step 3, to solve the MDP using value iteration, requires a number of iterations that is polynomial in $1/(1-\gamma), 1/\epsilon$, and the magnitude of the largest reward (Littman et al. 1995). Moreover, a single iteration of value iteration (Equation 7.7) can be performed in $O(|A||S|)$ steps.[7]

  For the single-agent case, $|S| = |\mathbf{T}|(\frac{\tau}{B} + 1)^{|\mathbf{C}|}$. The size of the action space $|A|$ depends on the connectedness of the bipartite graph $G_{\mathcal{C}}$, which is polynomial in the number of clusters. For the multi-agent case, for agent $\mathcal{A}_i$, $|S| = |\mathbf{T}|^i(\frac{\tau}{B}+1)^{|\mathbf{C}|}$. Its action space is identical to that of the single-agent MDP.

Thus, in both the single-agent and the multi-agent case, step 3 dominates the complexity of the algorithm; its computational complexity is exponential in the number of agents, as well as the number of clusters. This is in contrast with the algorithm proposed by Singh et al. (2009), which has polynomial complexity. It is interesting to investigate whether this difference stems from the fact that we compute infinite patrols, instead of finite ones. We intend to address this question in future work.

However, at this point it is worth mentioning that we can make considerable savings by disregarding states that are unreachable from initial state $\overline{s}$, as discussed in Section 7.2.1.3. We will empirically quantify these savings in Section 7.5. First, however, we study the use of online decentralised coordination to improve the multi-agent policy.

[7] For non-deterministic transition functions, value iteration needs $O(|A||S|^2)$ steps.

## 7.4 Improving the Multi-Agent Policy through Online Coordination

As mentioned earlier, the multi-agent policy that results from sequentially allocating agents is near-optimal. In this section, we investigate the use of two decentralised coordination techniques to ascertain whether it is possible to improve on these policies in an online phase. Using these coordination techniques, agents attempt to determine whether it pays off to deviate from their policies. In more detail, while being in states $s_1, \ldots, s_M$ agents attempt to find a *joint* action $\mathbf{a} = [a_1, \ldots, a_M]$ that yields a higher discounted reward than following policies $\pi_1, \ldots, \pi_M$:

$$\sum_{i=1}^{M} R(s_i, a_i) + \gamma^B V_{\pi_i}(\delta(s_i, a_1, \ldots, a_i)) > \sum_{i=1}^{M} R(s_i, \pi(s_i)) + \gamma^B V_{\pi_i}(\delta(s_i, \pi_1(s_1), \ldots, \pi_i(s_i))) \tag{7.13}$$

Computing a joint action that satisfies this equation raises a number of challenges. Firstly, the value functions $V_i$ have been computed only for those states $S_r(\overline{s})$ that are reachable from the initial state $\overline{s}$ (Equation 7.4), given that policies of agents $\mathbf{A}_{-i}$ are fixed. Thus, joint action $\mathbf{a}$ that deviates from these policies might cause several agents (with the notable exception of agent $\mathcal{A}_1$) to end up in a state $\hat{s} \notin S_r(\overline{s})$. Secondly, finding an action $\mathbf{a}$ that satisfies Equation 7.13 requires the evaluation of possibly many joint actions. As a result, evaluating Equation 7.13 for each of these actions can be very expensive.

To address the first challenge, we extend the state space $S_r(\overline{s})$ with the set $S_r(\hat{s})$ of those states that are reachable from the previously unreachable state $\hat{s}$. To compute the state values for each $s \in S_r(\hat{s}) \setminus S_r(\overline{s})$, we need to rerun the value iteration algorithm on these states only; the values of states $S_r(\overline{s})$ remain unaffected. This is because the value of a state depends on values of successors only (Equation 7.6), and states $S_r(\hat{s})$ are not successors of states $S_r(\overline{s})$ by definition.

To address the second challenge—minimising the computation overhead of finding a joint action that satisfies Equation 7.13—we investigate the use of two decentralised coordination approaches.

The first approach uses a more efficiently computable heuristic function $H$ that estimates the true value of a joint action $\mathbf{a}$ in state $\mathbf{s} = \{s_1, \ldots, s_M\}$. To goal then becomes to compute the action $\hat{\mathbf{a}}$ that maximises this function $H$:

$$\hat{\mathbf{a}} = \arg\max_{\mathbf{a}} H(\mathbf{s}, \mathbf{a}) \tag{7.14}$$

Since $H$ is an approximation of the value of an action, after computing $\hat{\mathbf{a}}$ agents need to determine whether it satisfies Equation 7.13, to ensure it improves the joint policy.

If it does, the action is chosen, if not, $\hat{\mathbf{a}}$ is discarded and the agents follow their original policies. To facilitate decentralised evaluation of the heuristic, we choose $H$ so it is decomposable into a sum of functions $H_i(s_i, a_i, \mathbf{a}_{-i})$—one for each agent—that depend on a (small) subset of $\mathbf{a}_{-i}$. We can then directly apply the max-sum algorithm to find $\hat{\mathbf{a}}$ in Equation 7.14 in a decentralised fashion.

In our empirical evaluation, we choose $H(\mathbf{s}, \mathbf{a})$ to be the immediate reward received by performing action $\mathbf{a}$. This reward depends only on the actions of those agents that can patrol $C$ simultaneously, and is therefore easily decomposable. The following example illustrates this:

**Example 7.7.** *Let agents $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ be located at transfer nodes $T_1$, $T_6$ and $T_7$ in Figure 7.2 respectively. In this case, heuristic function $H$ can be decomposed into three factors:*

$$H(\mathbf{a}, \mathbf{s}) = H_1(s_1, a_1) + H_2(s_2, a_1, a_2) + H_3(s_3, a_2, a_3)$$

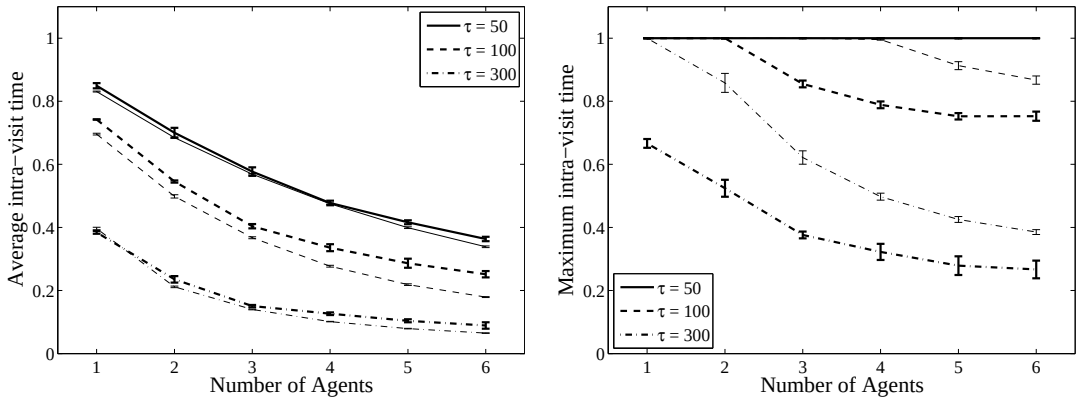
*This is because the immediate reward for agent $\mathcal{A}_1$ can not be affected by the other two agents, since it is not aware of their existence. In contrast, the reward for agent $\mathcal{A}_2$ can be affected by the actions of agent $\mathcal{A}_1$, if and only if agent 1 decides to patrol $C_2$. Similarly, the reward obtained by agent $\mathcal{A}_3$ is affected by agent $\mathcal{A}_2$ if the latter patrols $C_6$.*

The second approach involves the use of the Distributed Stochastic Algorithm (DSA) (see Section 2.6.1). The reason for choosing this algorithm is that it has a low computational complexity; the computation an agent needs to perform at each iteration is linear in the size of its action space. This is in contrast to max-sum, which has a complexity that is exponential in the number of neighbours. However, as mentioned in Section 2.6.1, DSA is also known to converge to poorer solutions than max-sum. Nevertheless, due to the inherent complexity of finding an action that satisfies Equation 7.13, a lesser computational demand is of decisive importance.

DSA works as follows. First, agents randomly initialise their actions $[a_1, \ldots, a_M]$. Then, in each iteration of the algorithm, $\mathcal{A}_i$ selects action $a_i^*$ as follows:

$$a_i^* = \arg\max_{a_i \in A(s)} \left[ R(s_i, a_i) + \gamma^B V_{\pi_i}(\delta(s_i, a_1, \ldots, a_i)) \right] \tag{7.15}$$

with respect to actions $a_1, \ldots, a_{i-1}$ being fixed. This continues for a fixed number of iterations (as in our empirical evaluation) or until convergence. While this algorithm is not guaranteed to yield an optimal solution, or even a solution that satisfies Equation 7.13, its main advantage is that it operates directly on the objective function in Equation 7.13, instead of a heuristic function. However, the main disadvantage is that it requires agent $\mathcal{A}_i$ to communicate with all agents $\mathbf{A}_{-i}$. This is due to the fact that—unlike the heuristic function used in the first approach—Equation 7.13 is not decomposable (which thus




(a) (b)

FIGURE 7.6: Intra-visit time over 1000 time steps. Thick lines correspond to the non-myopic algorithm. Thin lines correspond to the RHS algorithm from the previous chapter.

immediately precludes the use of max-sum), because of the intricate interdependencies between the agents' future rewards expressed by value function $V_{\pi_i}$.

In the next section we empirically evaluate these two decentralised coordination algorithms, as well as the algorithm developed in Section 7.2.

## 7.5 Empirical Evaluation

We present two sets of experiments. In the first set, the agents' goal is to minimise the time between observing each vertex of the layout graph. In the second set, the agents are tasked with monitoring a spatial phenomenon modelled by a GP as in Section 6.3.1.

In both experiments, we measure the average solution quality (i.e. average last visit time, or average RMSE), as well as the worst-case solution quality (i.e. maximum last visit time, and maximum RMSE). The latter metric is of key importance in safety-critical applications commonly found in disaster management and security domains, since it is a measure of the maximum risk involved in relying on the situational awareness the agents provide.

We benchmark the non-myopic algorithm against the receding horizon control algorithm developed in the previous chapter (which hereafter shall be referred to as the RHC algorithm).

### 7.5.1 Experiment 1: Minimising Intra-Visit Time

The first experiment is set in the IAM lab from Figure 7.1. We consider a scenario in which the value of observing a vertex is equal to the number of time steps that

has elapsed since it has last been observed, with a maximum of $\tau$ (clearly, this makes observations older than $\tau$ independent from observations made at the current time step). Thus, the agent's goal is to minimise the time between two successive observations of each vertex.

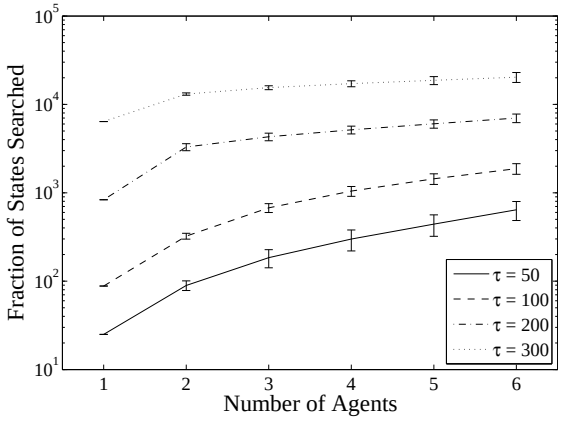
The agents have a circular observation area $\bigcirc_i$ with diameter $1.5m$.[8]

After initial calibration, we found that a budget $B$ of 50 leads to a partitioning of the graph in six clusters, such that agents are capable of observing all vertices within the allotted time of $B$. Step 1 of the algorithm identified the six clusters and seven transfer nodes shown in Figure 7.1. Vertices within a distance of $1/2\rho = .75$ (i.e. half the agents' observation radius) of vertices in different clusters were stripped away (not shown in Figure 7.1). We then applied the non-myopic algorithm with a varying number of agents.

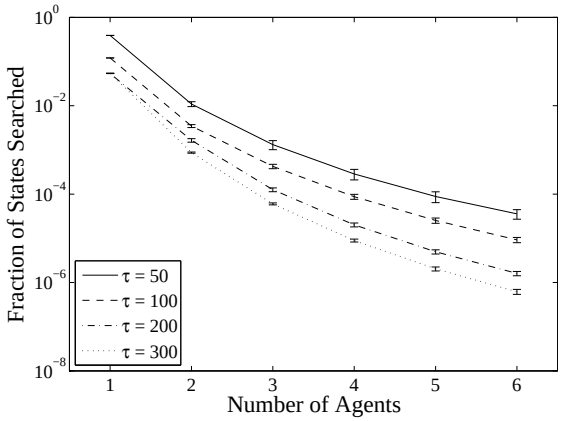
First, we analysed solution quality in terms of the average and maximum intra-visit time. To this end, we varied the temporal parameter $\tau$; the smaller this parameter, the quicker observations become stale, and the greater the need for an increased number of agents to accurately monitor the quicker changing environment. Results are shown in Figure 7.6 for 1000 time steps. From Figure 7.6(a), we can conclude that the average performance of our algorithm is comparable to that of the RHC algorithm, and that the incremental improvement of adding additional agents seems to flatten off more for our algorithm, specifically for higher values of $\tau$. The explanation for this is that the non-myopic algorithm patrols the graph in a more regular fashion, such that all clusters (and therefore all vertices) are visited in fixed intervals, while the RHC algorithm moves on immediately after the majority of value has been obtained. In terms of minimising the maximum intra-visit time, however, this behaviour is very effective, since the non-myopic algorithm reduces this time by approximately 30% for 6 agents compared to the RHC algorithm.

Second, we assessed the computational overhead of our algorithm. Figure 7.7(a) shows the number of states that were searched. This number is proportional to the running time of the value iteration algorithm (see Section 7.2.1.3), which represents the bulk of the total running time of our algorithm. This figure confirms that the number of states grows exponential in $\tau$, as expected. Furthermore, we found that the number of states is roughly linear in the number of agents, indicating that only a very small fraction of the exponentially large state space is reachable from the initial state. This is confirmed by Figure 7.7(b), which shows the size of the reachable state space $S_r(\bar{s})$ as a fraction of the $|\mathbf{T}|^M(\frac{\tau}{B}+1)^{|\mathbf{C}|}$ states (see Section 7.3.2). For six agents and $\tau = 300$ only one in $10^6$ states is reachable, and needs to be searched.

[8]Since the graph consists of lattice graphs in which the distance between adjacent vertices is 1m (Figure 7.3), an agent is capable of observing around 9 vertices simultaneously.




(a) Total number of reachable states.




(b) Reachable states as a fraction of all states.

FIGURE 7.7: The number of reachable states searched by the non-myopic algorithm.

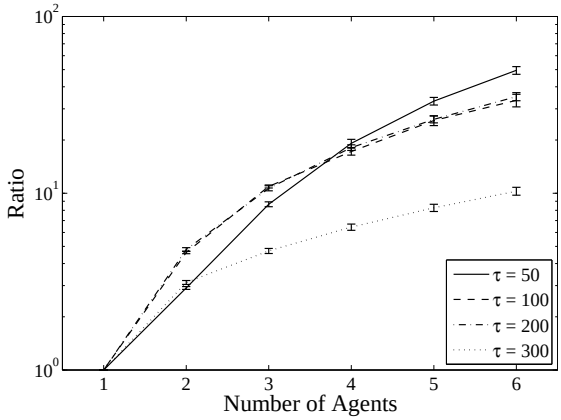



FIGURE 7.8: Fractional increase of the number of searched states using DSA.

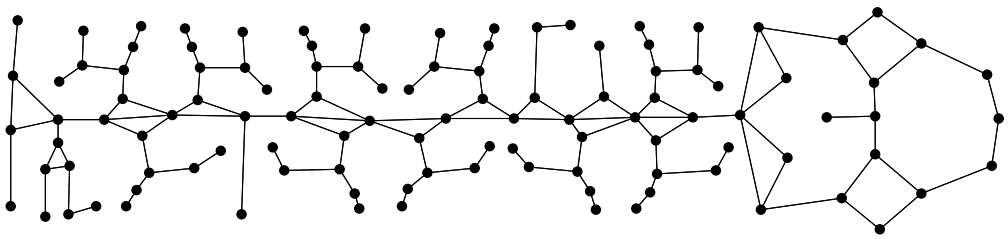


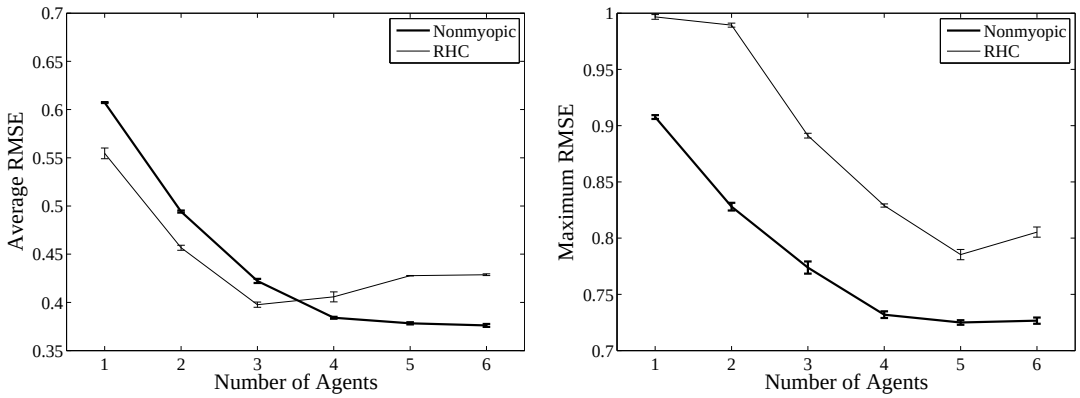
FIGURE 7.9: The ship layout graph used in Experiment 2.

Finally, we determined how much the computed policies can be improved by applying the two online coordination techniques discussed in Section 7.4. The first approach, based on maximising immediate reward, did not result in a statistically significant improvement of the joint policy ($< 1\%$). We can conclude from this that joint actions that maximise immediate reward are unlikely to maximise the sum of discounted rewards received by the agents, which means that the joint policy yields better results than a purely greedy policy. The second approach, which uses DSA, yielded an average improvement of 5%. Unfortunately, this improvement comes at a great cost. As discussed in Section 7.4, deviating from the policy leads to an increase in the number of states that need to be searched. Figure 7.8 shows the ratio between the states that were searched with and without coordination. From this figure, we can conclude that the size of the searched state space increases by two orders of magnitude to achieve this moderate improvement. Despite these negative results, we can consider the relative lack of effectiveness of both optimisation techniques as corroborating Lemma 7.2, which states that the policies computed by the sequential allocation method are close to optimal.

### 7.5.2 Experiment 2: Monitoring Environmental Phenomena

In this second experiment, the agents are tasked to patrol a ship (Figure 7.9) while monitoring an environmental phenomena. Thus, this experiment is similar to that described in Section 6.3.1. We used a GP to model this spatial phenomenon, which has a spatial length-scale of 5 and a temporal length-scale of 50. This corresponds to $\rho = 10$ (with $\epsilon < 0.01$) and $\tau = 100$ (with $\delta < 0.01$). These parameters were chosen to generate difficult problem instances. In these problem instances, the spatial phenomenon has a strong correlation along the temporal dimension (i.e. it varies slowly over time), and relatively weak correlations along the spatial dimension (i.e. it varies quickly in space). As a result, the agents' priority is to explore the space quickly, before settling into a patrolling routine that revisits vertices regularly.

The results in terms of average-case and worst-case RMSE are shown in Figure 7.10. The pattern observed here is similar to that of the first experiment: the non-myopic algorithm performs comparably to the RHC algorithm in terms of average-case performance, but outperforms it in terms of worst-case performance: the non-myopic algorithm reduces maximum RMSE by approximately 10% for 6 agents compared to the RHC algorithm.




(a) (b)

FIGURE 7.10: RMSE over 1000 time steps.

Moreover, while the marginal performance increase exhibited by the non-myopic algorithm is guaranteed to be positive (adding agents never hurts), the performance of the RHC algorithm starts to decline after adding the fourth agent.[9]

In summary, the empirical results show that our algorithm performs comparably to the receding horizon control algorithm developed in the previous chapter (which does not give performance guarantees, but has lower computational complexity) in terms of average performance, and significantly better in the worst-case. Furthermore, we found that improving upon the policies computed by this algorithm using online coordination is possible, but comes at a computational cost that is too high for the moderate improvement it yields. On the positive side, we consider this as evidence of the near-optimality of the joint policy as proved in Lemma 7.2.

## 7.6 Summary

In this chapter, we developed a non-myopic algorithm for computing infinitely long patrols for multiple mobile agents. This algorithm follows a similar three-step computation as the algorithm by Singh et al. (2009), i.e., decompose the environment into clusters, compute subpatrols within each cluster, and concatenate these subpatrols to form the desired patrol. However, unlike Singh et al. (2009), we consider environments that exhibit the property of *temporality*, which models a continuous rate of change. As a consequence, the algorithm proposed by Singh et al. (2009) computes finitely long paths, which tend not to return to previously visited locations, since no additional information (or value) can be obtained from doing so. In contrast, the patrols computed by our algorithm are

[9]This is caused by the max-sum algorithm that lies at the foundation of the RHS algorithm; as coordination between the agents becomes denser (i.e. agents need to coordinate with increasingly many neighbours), the factor graph contains more cycles causing max-sum to output increasingly less optimal solutions (Farinelli, Rogers, Petcu & Jennings 2008).

designed to monitor continuously changing environments, and thus periodically (and infinitely often) return to the same location to provide up-to-date situational awareness.

Just as with the algorithm developed in the previous chapter, this non-myopic algorithm operates on the generic problem definition in Chapter 3 and is thus applicable to a wide range of information gathering domains. We demonstrated this by analysing its performance in two distinct domains, and concluded that it performs comparably to the receding horizon control algorithm from the previous chapter (which has no performance guarantees, but lower computational complexity) in the average-case, but outperforms it in the worst-case.

Furthermore, we investigated the use of two decentralised coordination algorithms for improving the near-optimal multi-agent policy. Unfortunately, empirical evidence showed that neither technique is effective: the first has a low computational overhead, but yielded no discernible improvement, while the second moderately improves the solution (5%) at an unacceptably high computational cost. Despite this, we think it is unwise at this point to discard decentralised coordination as a means for improving policies computed by sequential allocation, and believe deeper study is required to unify these two valuable techniques. We come back to this issue in Section 8.2 where we discuss future work.

Now, in terms of the design requirements stated at the start of this thesis, we can conclude the following:

**Quality:** The empirical results from Section 7.5 show that it performs comparably to the receding horizon control algorithm developed in the previous chapter in terms of average-case solution quality, and $> 10\%$ better in terms of worst-case solution quality. We can therefore conclude that it is comparatively better able to provide situational awareness.

**Adaptiveness:** The policies computed by the non-myopic algorithm are not inherently adaptive. The occurrence of unexpected events that change observation value function $f$ therefore requires recomputation of these policies.

**Robustness:** The robustness of the algorithm depends on how policies are computed. This can be done in two ways: by a centralised controller or by the agents themselves. In the latter case, the sequential allocation method requires that agent $\mathcal{A}_i$ waits until agent $\mathcal{A}_{i-1}$ has computed its policy and transferred it to $\mathcal{A}_i$, thus creating a chain of computation. While this latter method is slightly more robust than a centralised controller (which is a single point of failure), it depends heavily on the existence of reliable communication channels. Thus, the non-myopic algorithm is less robust than the algorithm developed in the previous chapter, which is much less sensitive to unreliable communication.

**Autonomy:** Similar to robustness, the level of autonomy of the agents depends on the method of computation. The agents are not autonomous if a centralised controller is used to compute the policies. However, if policies are computed using the aforementioned computation chain, the algorithm allows agents to respond optimally to the policies of their predecessors, thus giving them autonomy over their actions.

**Scalability:** As discussed in Section 7.3.2, the complexity of both the single-agent and the multi-agent algorithm is exponential in the number of clusters, as well as the number of agents. This is in contrast with the algorithm developed by Singh et al. (2009), which has polynomial complexity. Further investigation is needed to ascertain whether this is inherent in the patrolling problem we considered in this chapter, and thus, whether performance guarantees and scalability are mutually exclusive for this problem.

**Modularity:** The same considerations as in Chapter 6 apply here, i.e. it is possible to define the action space of agents in such a way as to reflect their different capabilities, by having multiple different layout graphs $G$ for each type of agent.

**Performance Guarantees:** The algorithm developed in this chapter gives strong performance guarantees; in Section 7.3.1 we derived bounds on the observation value achieved by the single agent algorithm, as well as for the multi-agent algorithm.

# Chapter 8

# Conclusions and Future Work

In this chapter, we present an overview of the contributions of this thesis towards the research aim of decentralised coordination of teams of information gathering agents. First, in Section 8.1, we summarise the research carried out within each chapter. In so doing, we also ascertain the extend to which each contribution has satisfied the design requirements laid down at the beginning of the thesis. Then, in Section 8.2, we identify fruitful directions for future work that arise from our work.

## 8.1 Summary of Results

Unmanned sensors are rapidly becoming the *de facto* means of achieving situational awareness—the ability to make sense of and to predict what is happening in an environment—in disaster management, military reconnaissance, space exploration, and climate research. In these domains, and many others besides, their use reduces the need for exposing humans to hostile, impassable or polluted environments. Whilst these sensors are often currently pre-programmed or remotely controlled by human operators, there is a clear trend toward making these sensors fully autonomous, thus enabling them to make decisions without human intervention.

Full autonomy has two clear benefits over pre-programming and human remote control. First, in contrast to sensors with pre-programmed motion paths, autonomous sensors are better able to adapt to their environment, and react to *a priori* unknown external events or hardware failure. Second, autonomous sensors can operate in large teams that would otherwise be too complex to control by human operators. The key benefit of this is that a team of cheap, small sensors can achieve through cooperation the same results as individual large, expensive sensors—with more flexibility and robustness.

In light of the importance of autonomy and cooperation, we adopted an agent-based perspective on the operation of the sensors. Within this perspective, each sensor becomes

an *information gathering agent*. As a team, these agents can then direct their collective activity towards collecting information from their environment with the aim of providing accurate and up-to-date situational awareness.

Against this background, the central problem we addressed in this thesis is that of achieving accurate situational awareness through coordination of multiple information gathering agents. To achieve general and principled solutions to this problem, we formulated a generic problem definition, which captures the essential properties of dynamic environments. Specific instantiations of this generic problem span a broad spectrum of concrete application domains, of which we studied three canonical examples: monitoring environmental phenomena, wide area surveillance, and search and patrol.

The four main contributions of this thesis solve this general problem with additional constraints and requirements. First, in Chapters 4 and 5, we developed two decentralised coordination algorithms for settings with fixed agents that execute during the *deployment* (Chapter 3) and *operation* (Chapter 4) phase of the agents' life cycles. Then in Chapter 6 and 7, we developed two decentralised coordination algorithms for mobile agents: a scalable *receding horizon control* algorithm, and a *non-myopic* algorithm that provides performance guarantees on the quality of the achieved situational awareness.

In more detail, in Chapter 4, we studied the problem of maximising observation value, while simultaneously constructing a reliable communication network between the agents. Specifically, we considered the frequency allocation problem, which is equivalent to the NP-hard problem of graph colouring, and presented a novel approach that, rather than solving the graph colouring problem in the original network, deactivates certain agents, such that the communication graph that exists between the remaining agents is triangle-free, hence allowing the use of a simple decentralised graph colouring algorithm. We showed that this modified problem—maximising observation value subject to the communication graph being triangle-free—is also an NP-hard problem. We derived a centralised greedy and decentralised greedy approximation algorithm, and proved a $\frac{1}{7}$ approximation bound on the former. Empirical evidence showed that both algorithms perform close to optimal ($> 90\%$) and provide more than 250% more observation value over time compared to activating all agents simultaneously.

The second topic of investigation was decentralised coordination of fixed agents during the operational phase of their life cycle, and was described in Chapter 5. Specifically, we addressed the challenge of coordinating these agents under the assumption that their control parameters are continuous. In doing so, we developed two algorithms that extend the max-sum algorithm to continuous variables. The first algorithm, CPLF-MS, used techniques from computational geometry to derive exact algorithmic solutions for performing the two key mathematical operations required by max-sum for continuous piecewise linear functions. We benchmarked CPLF-MS against the standard max-sum

algorithm and a centralised simulated annealing algorithm, and found that it outperforms the former by up to 10%, and yields solutions close to the optimal solution computed by the latter. However, we also found that the complexity inherent in using simplexes to represent the utility functions tends to scale unfavourably with the number of neighbouring agents. The second algorithm, HCMS, avoids these problems. It uses non-linear optimisation techniques to evolve the variable domains used by the standard max-sum algorithm in settings with non-linear utility functions. We proved that the HCMS algorithm outperforms the standard max-sum algorithm, and, for sufficiently fine discretisations, the HCMS algorithm converges to a near optimal solution. Furthermore, empirical evidence shows that HCMS increases solution quality by up to 30% compared to the standard max-sum algorithm, at the cost of an at most threefold increase in the size of the messages. Moreover, with a sufficient number of iterations, it performs comparably to the near-optimal centralised simulated annealing algorithm.

Having dealt with fixed agents in Chapters 4 and 5, the second part of this thesis (Chapters 6 and 7) considered mobile agents, whose movements are restricted by their environment. In Chapter 6, we first developed an efficient receding-horizon algorithm, which coordinates the agents' movements so as to maximise the collective observation value received over a finite number of time steps in the future. To implement receding horizon control in a decentralised fashion, we again opted for the max-sum algorithm. However, due to the potentially very large action spaces of individual agents, the straightforward application of the max-sum algorithm to this problem was shown to be infeasible. Therefore, we made three augmentations to improve the scalability of our solution; all of which aim to reduce the action space that needs to be searched by the agents running the max-sum algorithm. The first exploits the property of *locality* of the observation value function to reduce the number of dependencies between agents, which results in an exponential reduction of the joint action space agents need to search. The second augmentation involves two heuristics for defining an individual agent's action space to reduce its action space. Both reduce an exponentially large set of possible paths to a small selection of paths that are likely to lead to good performance. The third augmentation involves two pruning techniques designed to speed up the max-sum algorithm itself. These techniques are general in the context of max-sum, and their application is thus not limited to the multi-agent information gathering problem we addressed in this thesis. We empirically evaluated the decentralised receding horizon algorithm in three different information gathering domains—environmental monitoring, pursuit-evasion and patrolling—and showed that it outperforms the state of the art by at least 30%. Most importantly, this algorithm is the first online decentralised algorithm for these these domains.

Following this, in Chapter 7, we developed a coordination algorithm for mobile agents with performance guarantees. Specifically, we developed a *non-myopic* algorithm for

computing a *patrol*—an infinitely long path for a single mobile agent designed to continuously monitor rapidly changing environments. This algorithm follows a similar three-step computation as the algorithm by Singh et al. (2009), i.e., decompose the environment into clusters, compute subpatrols within each cluster, and concatenate these subpatrols to form the desired patrol. However, the novelty of this algorithm lies in the fact that, unlike Singh et al., it considers environments that exhibit the property of *temporality*, which models a continuous rate of change. As a consequence, the algorithm proposed by Singh et al. computes finitely long paths, which tend not to return to previously visited locations, since no additional information (or value) can be obtained from doing so. In contrast, the patrols computed by our algorithm are designed to monitor continuously changing environments, and thus periodically (and infinitely often) return to the same location to provide up-to-date situational awareness. We subsequently extended this single-agent algorithm to the multi-agent case based on the method of *sequential allocation*. This method computes a near-optimal joint policy for a set of agents, by greedily computing single-agent policies conditioned on previously computed policies. We empirically evaluated both single-agent and multi-agent algorithms in two distinct domains, and concluded that it performs comparably to the receding horizon control algorithm from the previous chapter in the average-case, but outperforms it in the worst-case by $> 10\%$. Furthermore, we investigated the use of two decentralised coordination algorithms for improving the near-optimal multi-agent policy. Unfortunately, empirical evidence showed that neither technique is effective. The first has a low computational overhead, but yielded no discernible improvement, while the second moderately improves the solution (5%) at an unacceptably high computational cost. Despite this, we think it is unwise to discard decentralised coordination as a means for improving policies computed by sequential allocation, and believe deeper study is required to unify these two valuable techniques. We come back to this issue in Section 8.2 where we discuss future work.

More specifically, looking back at the design requirements identified at the start of this thesis, we can conclude that we successfully addressed each of them:

**Quality:** For each algorithm developed in this thesis, we have given extensive empirical evidence of the achieved quality of situational awareness, either by comparing them against the state of the art, or against optimal (centralised) algorithms.

**Adaptiveness:** The online coordination algorithms for fixed agents (Chapter 5) and the receding horizon control algorithm for mobile agents (Chapter 6) can run continuously during the operation of the agents, and are thus capable of adapting to changes in their environment. In contrast, the coordination algorithms for deploying fixed agents (Chapter 5) and the non-myopic algorithm for mobile agents (Chapter 7) both have an offline computation phase. While it is possible to revise solutions after the occurrence of *a priori* unknown events, this requires

re-computation of the solution. In case of the former algorithm, the cost associated with doing this is limited (since its computational complexity is polynomial). However, in case of the latter, agents have to compute new policies, which takes an exponential number of computation steps (as a function of the number of agents) in the worst case. Thus, while the algorithm from Chapter 5 can be regarded as adaptive if run periodically, the non-myopic algorithm from Chapter 7 cannot.

**Robustness:** All algorithms in this thesis, with the exception of the non-myopic algorithm for mobile agents (Chapter 7), are true multi-agent solutions. As such, control is divided over multiple agents, and thus no central point of failure exists. This is somewhat in contrast to the non-myopic algorithm for multiple agents, which creates a chain of computation. This means that agent $\mathcal{A}_i$ has to wait until agent $\mathcal{A}_{i-1}$ has computed its policy and transferred it to $\mathcal{A}_i$. Thus, this algorithm is merely as robust as the weakest link in this chain.

**Autonomy:** All algorithms enable the agents to make decisions without the intervention of a human operator or a central controller. So, we can conclude that the agents operate autonomously.

**Scalability:** All algorithms in this thesis, except for the non-myopic algorithm for mobile agents (Chapter 7), exhibit a computational overhead that scales with the number of neighbours only—not with the size of the team. The decentralised algorithm for deploying fixed agents (Chapters 4) was shown to be very scalable, since the computation performed by a single agent increases polynomially with the number of neighbours, which is much less than the number of agents in the system as a whole. Similarly, the algorithms for coordinating fixed agents (Chapter 5) and receding horizon control of mobile agents (Chapter 5) have a computational overhead that scales with the number of neighbours (albeit exponentially).

**Modularity:** Using the central problem formulation in Chapter 3, we demonstrated that it is possible to model agents with heterogeneous sensing capabilities (e.g. in terms of sensing accuracy or observation area), and movement capabilities and constraints (e.g. speed, ground based or airborne). Since the algorithms developed in this thesis operate directly on this problem formulation, they support teams of heterogeneous agents by design, and as such, do not impose constraints on the implementation of the sensing platforms in which these agents are embedded. Moreover, the CPLF-MS and HCMS algorithms for continuous decentralised coordination problems developed in Chapter 5 further increase modularity, by allowing agents whose controls are inherently continuous to inter-operate, without the need for artificial discretisation of their control variables.

**Performance Guarantees:** Both the centralised greedy algorithm for deploying fixed agents from Chapter 4, and the non-myopic multi-agent patrolling algorithm from Chapter 7 give performance guarantees.

Thus, when taken together, the contributions presented in this thesis represent a significant advance in the state of the art of decentralised coordination of information gathering agents. However, many open problems remain.

## 8.2 Future Work

Despite these accomplishments, there are still a number of open issues to be addressed. Whilst the contributions were successful in addressing all of the design requirements laid down at the beginning of this thesis, taken individually they satisfy (strict) subsets of these requirements. Specifically, our contributions seem to suggest a trade-off between performance guarantees on the one hand, and scalability, adaptivity, and robustness on the other hand (see Table 1.2). Thus, we identify a need to combine different aspects of these algorithms to simultaneously satisfy all requirements.

To this end, we believe further investigation is needed to simultaneously achieve the performance guarantees of the non-myopic algorithm in Chapter 7 and the scalability, robustness and adaptivity of the receding horizon algorithm in Chapter 6. In more detail, we are particularly interested in combining decentralised coordination with non-myopic online performance guarantees. The use of decentralised coordination to improve on the near-optimal multi-agent policy in Chapter 7 can be considered as an initial attempt to achieve this. Here, the coordination process was guided by the non-myopic state values resulting in high-quality long term solutions (albeit against an unacceptably high computational overhead). Thus, future work should concentrate on addressing this issue. One promising direction of investigation could focus on replacing the (intrinsically less-scalable) MDP formulated in Chapter 7 by the efficient (i.e. polynomial) modular orienteering algorithm by Checkuri & Pal (2008). This algorithm lies at the foundation of the algorithm proposed by Singh et al. (2009) for computing finitely long paths of high value, and solves a special case of the problem we formulated in chapter 3, in which all observations are independent. However, to suit our purposes, this algorithm must be extended to compute cycles instead of walks, and support the property of *temporality*.

Beyond this topic for future research that stems directly from the work presented in this thesis, we further identify three specific lines of investigation to extend the scope and applicability of our work:

**Multi-Objective Information Gathering** The central problem formulation in Chapter 1 states that agents should seek to maximise the quality of their situational awareness through coordination. No mention is made of whether this pertains to single or multiple phenomena of interest. More specifically, in this thesis we have addressed the challenge of maximising situational awareness for a *single* feature, be this the value of an environmental phenomenon, the location of evader, or the class

of a target. However it is conceivable, and even likely, that there exist multiple features that need to be monitored or found simultaneously (e.g. fire, poisonous gas, wounded civilians, etc.). Moreover, these features might require the agents to perform different, mutually exclusive actions (e.g. civilians are located in one part of a building, while a fire is raging in another). Simply weighing these objectives can lead to situations in which one feature is being monitored with high quality, while others are being ignored.

One area to look for potential insights to avoid such situations is multi-objective optimisation, which seeks to simultaneously optimise two or more conflicting objectives (Steuer 1986). The solution to a multi-objective problem is a set of Pareto-optimal solutions—solutions in which it is impossible to further satisfy one objective without reducing the solution quality of others. Within this set, a solution can be chosen that satisfies certain constraints, such as maximising the quality of situational awareness for one feature, subject to a minimum quality for another. At the moment of this writing, a decentralised algorithm for multi-objective optimisation is under development in our lab. This algorithm is based on max-sum, which has been shown throughout this thesis to be very suited to the coordination of information gathering agents, and is thus an obvious point of departure.

**Adversarial Domains** In this thesis, we have considered environments that are non-strategic, i.e. these environments do not behave so as to further their interest, since they simply had none, or were assumed to have none. However, there are two reasons why this assumption should be dropped for some classes of applications. Firstly, some scenarios are clearly intrinsically strategic, such as those found in the pursuit evasion and patrolling domain. In Chapter 6 these scenarios were characterised by attackers with a probabilistic behaviour model, rather than an *adversarial* strategic one, and thus they did not behave in such a way so as to minimise the probability of their capture. Secondly, assuming the environment behaves strategically—even when it does not—is equivalent to being fully risk averse, in the sense that good solutions to this problem seek to minimise the maximum risk the agents (and their owners) are exposed to. In safety-critical and hostile scenarios, this is clearly a desirable trait.

Krause et al. (2008) have studied the related problem of minimising the maximum vulnerability of a water distribution network to attacks with contaminants by placing sensors at key locations, and demonstrated that this metric is submodular. Thus, modifying the problem definition in Chapter 3 to strategic environments should be straightforward, although further investigation is required to determine whether patrolling in the presence of *moving* strategic intruders (Agmon, Kraus & Kaminka 2008) can also be captured in this framework. This modification makes it possible to apply all of the algorithms in this thesis in their existing form, with one exception. With these aforementioned 'mini-max' submodular functions, the

receding-horizon control algorithm from Chapter 6 might not be able to distinguish between good and bad actions, since many paths of finite length are equivalent in their ability to reduce maximum risk. This is because the (relatively) short paths over which the agents coordinate are not likely arrive at those locations where this risk is currently maximal. Note that the non-myopic algorithm from Chapter 7 avoids this problem because it considers infinitely long paths, but this approach scales unfavourably with the number of agents. Thus, this strand of research has some overlap with the aforementioned investigation into efficient hybrid non-myopic/decentralised coordination solutions.

**Anytime Algorithms:** One of the common properties of the algorithms developed in this thesis is that action planning and execution take place in distinct phases. In the first phase, agents coordinate to maximise observation value received as a team, after which the resulting coordinated plan is implemented in the second phase. However, in some cases, time constraints require one or more agents to terminate the coordination algorithm prior to completion. For instance, a sudden unexpected event might require an immediate response, or an uneven distribution of the computational workload might cause agents to time-out waiting for a response from a neighbour.[1] In such cases, it is imperative that the agents are not left without a solution—a situation which can have detrimental effects, such as the failure of one or more agents, or a severe reduction in situational awareness. To this end, an investigation into *anytime* coordination algorithms could improve both the *quality* of situational awareness as well as the *adaptiveness* of a team of information gathering agents. The key property of an anytime algorithm is its capability to compute partial solutions, the quality of which is non-decreasing in the amount of computation the algorithm was able to perform before termination.

One interesting line of research in this area could seek to extend the max-sum algorithm. This, however, creates a non-trivial challenge, for the max-sum algorithm (and, indeed, decentralised computation in general) is asynchronous by nature. As a result, computation can have been progressed further in agents that are assigned a small workload, than in those that are burdened by a larger one. Thus, when termination is triggered by one of the agents, the impartial solution computed by the algorithm is likely to be out-of-sync, leading to uncoordinated (and thus poor) behaviour. This problem adds additional difficulty to the fact that, before the max-sum algorithm has converged, it can yield arbitrarily poor solutions. Nevertheless, a modification to the max-sum algorithm that allows it to compute increasingly better solutions over time could be a significant contribution to the field of decentralised coordination in general, and decentralised coordination of information gathering agents in particular.

[1] In general, an agent whose utility depends on many neighbours needs to perform more computation than an agent with few neighbours. For example, in the max-sum algorithm, the amount of computation an agent needs to perform scales exponentially with the number of neighbours.

By meeting these challenges, the practical applicability of decentralised coordination algorithms developed in this thesis can be further increased, which will bring the true autonomous control of unmanned sensors one step closer.

# Bibliography


Agmon, N., Kraus, S. & Kaminka, G. A. (2008), Multi-robot perimeter patrol in adversarial settings, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)', pp. 2339–2345.

Agmon, N., Sadov, V., Kaminka, G. A. & Kraus, S. (2008), The impact of adversarial knowledge on adversarial planning in perimeter patrol, *in* 'Proceedings of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', Estoril, Portugal, pp. 55–62.

Ahmadi, M. & Stone, P. (2006), A multi-robot system for continuous area sweeping tasks, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)', pp. 1724–1729.

Aji, S. M. & McEliece, R. J. (2000), 'Generalized distributive law', *IEEE Transactions on Information Theory* **46**(2), 325–343.

Avriel, M. (1999), *Nonlinear Programming: Analysis and Methods*, Dover Publishing.

Basilico, N., Gatti, N. & Amigoni, F. (2009), Leader-follower strategies for robotic patrolling in environments with arbitrary topologies, *in* 'Proceedings of the Eighth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', Budapest, Hungary, pp. 57–64.

Bian, F., Kempe, D. & Govindan, R. (2006), Utility-based sensor selection, *in* 'Proceedings of the Fifth International Conference on Information Processing in Sensor Networks (IPSN)', Vol. 2006, Nashville, TN, United States, pp. 11–18.

Bollobás, B., Catlin, P. A. & Erdõs, P. (1980), 'Hadwiger's conjecture is true for almost every graph', *European Journal on Combinatorics* **1**, 195–199.

Bopardikar, S. D., Bullo, F. & Hespanha, J. P. (2008), A pursuit game with range-only measurements, *in* 'Proceedings of the Forty-Seventh Conference on Decision and Control', pp. 4233–4238.

Borie, R., Tovey, C. & Koenig, S. (2009), Algorithms and complexity results for pursuit-evasion problems, *in* 'Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)', Pasadena, California, USA, pp. 59–66.

Bryan, K. L., Ren, T., DiPippo, L., Henry, T. & Fay-Wolfe, V. (2007), Towards optimal TDMA frame size in wireless sensor networks, Technical report, University of Rhode Island.

Calinescu, G., Chekuri, C., Pál, M. & Vondrák, J. (2007), Maximizing a submodular set function subject to a matroid constraint (extended abstract), *in* 'Proceedings of the Twelfth International Conference on Integer Programming and Combinatorial Optimization', pp. 182–196.

Calisi, D., Farinelli, A., Iocchi, L. & Nardi, D. (2007), Autonomous exploration for search and rescue robots, *in* 'WIT Transactions on the Built Environment', Vol. 94, Malta, pp. 305–314.

Checkuri, C. & Pal, M. (2008), Improved algorithms for orienteering and related problems, *in* 'Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)', pp. 661–670.

Christofides, N. (1976), Worst-case analysis of a new heuristic for the travelling salesman problem, Technical report, Carnegie Mellon University, Graduate School of Industrial Administration.

Dang, V. D., Dash, R. K., Rogers, A. & Jennings, N. R. (2006), Overlapping coalition formation for efficient data fusion in multi-sensor networks, *in* 'Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)', Boston, Massachusetts, USA, pp. 635–640.

Dash, R. K., Rogers, A., Jennings, N. R., Reece, S. & Roberts, S. (2005), Constrained bandwidth allocation in multi-sensor information fusion: a mechanism design approach, *in* 'Proceedings of the Seventh International Conference on Information Fusion (FUSION)', Philadelphia, Pennsylvania, USA.

Deshpande, A., Guestrin, C. & Madden, S. (2005), 'Resource-aware wireless sensor-actuator networks', *IEEE Data Engineering Bulletin* **28**(1), 40–47.

Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J. M. & Hong, W. (2005), 'Model-based approximate querying in sensor networks', *International Journal on Very Large Data Bases* **14**(4), 417–443.

Dunias, P. (1996), Robot motion planning using potential fields, *in* 'Proceedings of the Symposium on Robotics and Cybernetics. CESA '96 IMACS Multiconference. Computational Engineering in Systems Applications', Lille, France, pp. 611–616.

Durrant-Whyte, H. F., Rao, B. Y. S. & Hu, H. (1990), Toward a fully decentralized architecture for multi-sensor data fusion, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)', Vol. 2, Los Alamitos, California, USA, pp. 1331–1336.

Dvořák, Z., Kawarabayashi, K. & Thomas, R. (2009), Three-coloring triangle-free planar graphs in linear time, *in* 'Proceedings of the ACM-SIAM Symposium on Discrete Algorithms', pp. 1176–1182.

Edachery, J., Sen, A. & Brandenburg, F. (1999), Graph clustering using distance-k cliques, *in* J. Kratochvyl, ed., 'Graph Drawing', Vol. 1731 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 98–106.

Edordu, C. & Sacks, L. (2006), Self organising wireless sensor networks as a land management tool in developing countries: A preliminary survey, Technical report, University College London.

Endsley, M. R. (1995), 'Toward a theory of situation awareness in dynamic systems', *Human Factors* **37**(1), 32–64.

Farinelli, A., Rogers, A. & Jennings, N. R. (2008), Maximising sensor network efficiency through agent-based coordination of sense/sleep schedules, *in* 'Workshop on Energy in Wireless Sensor Networks in conjuction with DCOSS 2008', pp. 43–56.

Farinelli, A., Rogers, A. & Jennings, N. R. (2009), Bounded approximate decentralised coordination using the max-sum algorithm, *in* 'Proceedings of the IJCAI Workshop on Distributed Constraint Reasoning (DCR)', Pasadena, California, USA, pp. 46–59.

Farinelli, A., Rogers, A., Petcu, A. & Jennings, N. R. (2008), Decentralised coordination of low-power embedded devices using the max-sum algorithm, *in* 'Proceedings of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', Estoril, Portugal, pp. 639–646.

Fitzpatrick, S. & Meertens, L. (2003), Distributed coordination through anarchic optimization, *in* V. Lesser, C. L. Ortiz, Jr. & M. Tambe, eds, 'Distributed Sensor Networks', Kluwer Academic Publishers, chapter 11, pp. 257–295.

Frey, B. J. & Dueck, D. (2007), 'Clustering by passing messages between data points', *Science* **315**(5814), 972–976.

Gallager, R. G., Humblet, P. A. & Spira, P. M. (1983), 'A distributed algorithm for minimum-weight spanning trees', *ACM Transactions on Programming Languages and Systems* **5**(1), 66–77.

Galmes, S. (2006), Lifetime issues in wireless sensor networks for vineyard monitoring, *in* 'Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)', Vancouver, Canada, pp. 542–545.

Garnett, R., Osborne, M., Reece, S., Rogers, A. & Roberts, S. J. (2009), Sequential bayesian prediction in the presence of changepoints, *in* 'Proceedings of the Twenty-Sixth Annual International Conference on Machine Learning (ICML)', Montreal, Quebec, Canada, pp. 345–352.

Giusti, R., Murphy, A. L. & Picco, G. P. (2007), Decentralized scattering of wake-up times in wireless sensor networks, *in* 'Proceedings of the Fourth European Conference on Wireless Sensor Networks', pp. 245–260.

Granville, V., Krivánek, M. & Rasson, J. (1994), 'Simulated annealing: A proof of convergence', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**, 652–656.

Grocholsky, B. (2002), Information-Theoretic Control of Multiple Sensor Platforms, PhD thesis, University of Sydney.

Grocholsky, B., Keller, J., Kumar, V. & Pappas, G. (2006), 'Cooperative air and ground surveillance', *IEEE Robotics & Automation Magazine* **13**(3), 16–25.

Grocholsky, B., Makarenko, A., Kaupp, T. & Durrant-Whyte, H. F. (2003), Scalable control of decentralised sensor platforms, *in* 'Proceedings of the Second International Workshop on Information Processing in Sensor Network (IPSN)', Palo Alto, California, USA, pp. 96–112.

Grocholsky, B., Swaminathan, R., Keller, J., Kumar, V. & Pappas, G. (2005), Information driven coordinated air-ground proactive sensing, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)', Barcelona, Spain, pp. 2211–2216.

Gross, J. & Yellen, J. (1999), *Graph theory and its applications*, CRC Press, Inc., Boca Raton, Florida, USA.

Guestrin, C., Krause, A. & Singh, A. P. (2005), Near-optimal sensor placements in gaussian processes, *in* 'Proceedings of the Twenty-Second International Conference on Machine Learning (ICML)', pp. 265–272.

Halvorson, E., Conitzer, V. & Parr, R. (2009), Multi-step multi-sensor hider-seeker games, *in* 'Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)', Pasadena, California, USA, pp. 159–166.

Hart, J. K., Martinez, K., Ong, R., Riddoch, A., Rose, K. C. & Padhy, P. (2006), 'A wireless multi-sensor subglacial probe: Design and preliminary results', *Journal of Glaciology* **51**(178), 389–397.

Hershberger, J. (1989), 'Finding the upper envelope of $n$ line segments in $O(n \log n)$ time', *Information Processing Letters* **33**(4), 169–174.

Hespanha, J. P., Kim, H. J. & Sastry, S. (1999), Multiple-agent probabilistic pursuit-evasion games, *in* 'Proceedings of the Thirty-Eighth Conference on Decision and Control', Vol. 3, pp. 2432–2437.

Howard, R. A. (1960), *Dynamic Programming and Markov Processes.*, The MIT Press, Cambridge, Massachusetts, USA.

Julier, S. & Uhlmann, J. (1997), A non-divergent estimation algorithm in the presence of unknown correlations, *in* 'Proceedings of the American Control Conference (ACC)', Vol. 4, pp. 2369–2373.

Kansal, A., Hsu, J., Zahedi, S. & Srivastava, M. B. (2007), 'Power management in energy harvesting sensor networks', *ACM Transactions on Embedded Computing Systems* **6**(4), article no. 32.

Karp, R. (1972), Reducibility among combinatorial problems, *in* R. Miller & J. Thatcher, eds, 'Complexity of Computer Computations', Plenum Press, pp. 85–103.

Kato, Y. & Mukai, T. (2005), A new method of localizing gas source positions for a mobile robot with gas sensors, *in* 'Proceedings of the Thirteenth International Conference on Solid-State Sensors, Actuators and Microsystems (TRANSDUCERS)', Seoul, South Korea, pp. 2155–2158.

Kawarabayashi, K. & Toft, B. (2005), 'Any 7-chromatic graphs has $K_7$ or $K_{4,4}$ as a minor', *Combinatorica* **25**, 327–353.

Kerr, W. & Spears, D. (2005), Robotic simulation of gases for a surveillance task, *in* 'Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems', Edmonton, Alberta, Canada, pp. 2905–2910.

Kho, J., Rogers, A. & Jennings, N. R. (2009), 'Decentralised control of adaptive sampling in wireless sensor networks', *ACM Transactions on Sensor Networks* **5**(3).

Kiekintveld, C., Yin, Z., Kumar, A. & Tambe, M. (2010), Asynchronous algorithms for approximate distributed constraint optimization with quality bounds, *in* 'Proceedings of the Ninth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', Toronto, Canada, pp. 133–140.

Kim, Y., Krainin, M. & Lesser, V. (2010), Application of Max-Sum Algorithm to Radar Coordination and Scheduling, *in* 'Proceedings of the Twelfth International Workshop on Distributed Constraint Reasoning', Toronto, Canada.

Ko, C. W., Lee, J. & Queyranne, M. (1995), 'An exact algorithm for maximum entropy sampling', *Operations Research* **43**(4), 684–691.

Krause, A. & Guestrin, C. (2005), Near-optimal nonmyopic value of information in graphical models, *in* 'Proceedings of the Twenty-First Annual Conference on Uncertainty in Artificial Intelligence (UAI)', Edinburgh, Scotland, UK, pp. 324–331.

Krause, A. & Guestrin, C. (2007), Nonmyopic active learning of gaussian processes: an exploration-exploitation approach, *in* 'Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML)', ACM Press, New York, New York, USA, pp. 449–456.

Krause, A., Guestrin, C., Gupta, A. & Kleinberg, J. (2006), Near-optimal sensor placements: Maximizing information while minimizing communication cost, *in* 'Proceedings of the Fifth International Conference on Information Processing in Sensor Networks (IPSN)', ACM Press, New York, NY, USA, pp. 2–10.

Krause, A., Leskovec, J., Guestrin, C., VanBriesen, J. & Faloutsos, C. (2008), 'Efficient sensor placement optimization for securing large water distribution networks', *Journal of Water Resources Planning and Management* **134**(6), 516–526.

Kschischang, F. R., Frey, B. J. & Loeliger, H. A. (2001), 'Factor graphs and the sum-product algorithm', *IEEE Transactions on Information Theory* **47**(2), 498–519.

Langendoen, K., Baggio, A. & Visser, O. (2006), Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture, *in* 'Proceedings of the Fourteenth International Symposium on Parallel and Distributed Real-Time Systems (WPDRTS)', Rhodes, Greece, pp. 155–162.

Lilienthal, A., Reiman, D. & Zell, A. (2003), Gas source tracing with a mobile robot using an adapted moth strategy, *in* 'Proceedings of the Autonomous Mobile Systems (AMS)', Stuttgart, Germany, pp. 150–160.

Littman, M. L., Dean, T. L. & Kaelbling, L. P. (1995), On the complexity of solving Markov Decision Problems, *in* 'Proceedings of the Eleventh International Conference on Uncertainty in Artificial Intelligence (UAI)', Montreal, Quebec, Canada, pp. 394–402.

Low, K. H., Dolan, J. M. & Khosla, P. (2008), Adaptive multi-robot wide-area exploration and mapping, *in* 'Proceedings of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', Estoril, Portugal, pp. 23–30.

MacKay, D. J. C. (2003), *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press.

Maheswaran, R. J., Pearce, J. & Tambe, M. (2005), A family of graphical-game-based algorithms for distributed constraint optimization problems, *in* 'Coordination of Large-Scale Multiagent Systems', Springer-Verlag, Heidelberg Germany, pp. 127–146.

Mailler, R. & Lesser, V. (2008), Solving distributed constraint optimization problems using cooperative mediation, *in* 'Proceedings of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', Estoril, Portugal, pp. 438–445.

Makarenko, A. & Durrant-Whyte, H. (2006), 'Decentralized bayesian algorithms for active sensor networks', *Information Fusion* **7**(4), 418–33.

Martinez-Cantin, R., de Freitas, N., Doucet, A. & Castellanos, J. A. (2007), Active policy learning for robot planning and exploration under uncertainty, *in* 'Proceedings of Robotics: Science and Systems'.

Meliou, A., Krause, A., Guestrin, C. & Hellerstein, J. M. (2007), Nonmyopic informative path planning in spatio-temporal models, *in* 'Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI)', Vancouver, British Columbia, Canada, pp. 602–607.

Modi, J. P., Scerri, P., Shen, W. & Tambe, M. (2003), Distributed resource allocation, *in* V. Lesser, C. L. Ortiz, Jr. & M. Tambe, eds, 'Distributed Sensor Networks', Kluwer Academic Publishers, chapter 10, pp. 219–256.

Modi, J. P., Tambe, M. & Yokoo, M. (2005), 'ADOPT: Asynchronous distributed constraint optimization with quality guarantees', *Artificial Intelligence* **161**, 149–180.

Moore, A. W. & Atkeson, C. G. (1993), 'Prioritized sweeping: Reinforcement learning with less data and less time', *Machine Learning* **13**(1), 103–130.

Moran, S. (1984), 'On the length of optimal TSP circuits in sets of bounded diameter', *Journal of Combinatorial Theory, Series B* **37**(2), 113 – 141.

Murphy, R., Casper, J., Hyams, J., Micire, M. & Minten, B. (2000), Mobility and sensing demands in USAR, *in* 'Proceedings of the Twenty-Sixth Annual Conference of the IEEE Industrial Electronics Society (IECON)', Vol. 1, Nagoya, Japan, pp. 138–142.

Nemhauser, G. L. & Wolsey, L. A. (1978), 'An analysis of approximations for maximising submodular set functions—I', *Mathematical Programming* **14**(1), 265—294.

Nemhauser, G. L., Wolsey, L. A. & Fisher, M. L. (1978), 'An analysis of approximations for maximizing submodular set functions—II', *Mathematical Programming Studies* **8**, 73–87.

Osborne, M. A., Rogers, A., Ramchurn, S. D., Roberts, S. J. & Jennings, N. R. (2008), Towards real-time information processing of sensor network data using computationally efficient multi-output gaussian processes, *in* 'Proceedings of the Seventh International Conference on Information Processing in Sensor Networks (IPSN)', St. Louis, Missouri, USA, pp. 109–120.

Padhy, P., Dash, R. K., Martinez, K. & Jennings, N. R. (2006), A utility-based sensing and communication model for a glacial sensor network, *in* 'Proceedings of the Fifth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', Hakodate, Japan, pp. 1353–1360.

Padhy, P., Dash, R. K., Martinez, K. & Jennings, N. R. (2010), 'A utility-based adaptive sensing and multi-hop communication protocol for wireless sensor networks', *ACM Transactions on Sensor Networks* **6**(3), article no. 27.

Parsons, T. (1978), 'Pursuit-evasion in a graph', *Theory and Application of Graphs* **642**, 426–441.

Paruchuri, P., Pearce, J., Tambe, M., Ordonez, F. & Kraus, S. (2007), An efficient heuristic approach for security against multiple adversaries, *in* 'Proceedings of the Sixth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', Honolulu, Hawaii, USA, pp. 1–8.

Paskin, M., Guestrin, C. & McFadden, J. (2005), A robust architecture for distributed inference in sensor networks, *in* 'Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks (IPSN)', pp. 55–62.

Pereira, G. A. S., Soares, M. B. & Campos, M. F. M. (2004), A potential field approach for collecting data from sensor networks using mobile robots, *in* 'Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', Vol. 4, Sendai, Japan, pp. 3469–3474.

Petcu, A. & Faltings, B. (2005), A scalable method for multiagent constraint optimization, *in* 'Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)', Edinburgh, Scotland, UK, pp. 266–271.

Pon, R., Kansal, A., Liu, D., Rahimi, M., Shirachi, L., Kaiser, W. J., Pottie, G. J., Srivastava, M., Sukhatme, G. & Estrin, D. (2005), Networked Infomechanical Systems (NIMS): Next generation sensor networks for environmental monitoring, *in* 'IEEE MTT-S International Microwave Symposium Digest', Vol. 2005, Long Beach, California, United States, pp. 373–376.

Puterman, M. L. (1994), *Markov Decision Processes—Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Inc., New York, New York, USA.

Puterman, M. L. & Shin, M. C. (1978), 'Modified policy iteration algorithms for discounted markov decision problems', *Management Science* **24**(11), 1127–1137.

Rahimi, M., Pon, R., Kaiser, W. J., Sukhatme, G. S., Estrin, D. & Srivastava, M. (2004), Adaptive sampling for environmental robotics, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)', Vol. 4, New Orleans, Louisiana, USA, pp. 3537–3544.

Rasmussen, C. E. & Ghahramani, Z. (2003), Bayesian Monte Carlo, *in* S. T. Suzanna Becker & K. Obermayer, eds, 'Advances in Neural Information Processing Systems', Vol. 15, MIT Press, pp. 489–496.

Rasmussen, C. E. & Williams, C. K. I. (2006), *Gaussian Processes for Machine Learning*, The MIT Press.

Reece, S. & Roberts, S. (2005), Robust, low-bandwidth, multi-vehicle mapping, *in* 'Proceedings of the Eighth International Conference on Information Fusion (Fusion 2005)', Vol. 2.

Reece, S. & Roberts, S. (2008), Information fusion and inference in dynamic inference environments, Technical Report D1.1-11, Oxford University, Department of Engineering Science.

Robertson, N., Seymour, P. & Thomas, R. (1993), 'Hadwiger's conjecture for $K_6$-free graphs', *Combinatorica* **13**, 279–361.

Rogers, A., Corkill, D. D. & Jennings, N. R. (2009), 'Agent technologies for sensor networks', *IEEE Intelligent Systems* **24**(2), 13–17.

Sak, T., Wainer, J. & Goldenstein, S. K. (2008), Probabilistic multiagent patrolling, *in* 'Proceedings of the Brazilian Symposium on Artificial Intelligence (SBIA)', pp. 124–133.

Schaeffer, S. E. (2007), 'Graph clustering', *Computer Science Review* **1**(1), 27–64.

Singh, A., Krause, A., Guestrin, C., Kaiser, W. J. & Batalin, M. A. (2007), Efficient planning of informative paths for multiple robots, *in* 'Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)', Hyderabad, India, pp. 2204–2211.

Singh, A., Krause, A. & Kaiser, W. J. (2009), Nonmyopic adaptive informative path planning for multiple robots, *in* 'Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)', Pasadena, California, USA, pp. 1843–1850.

Steuer, R. E. (1986), *Multiple Criteria Optimization: Theory, Computations, and Application*, John Wiley & Sons, Inc., New York, New York, USA.

Stranders, R., Delle Fave, F. M., Rogers, A. & Jennings, N. R. (2010), A decentralised coordination algorithm for mobile sensors, *in* 'Proceedings of the Twenty-Fourth National Conference on Artificial Intelligence (AAAI)', Atlanta, Georgia, USA, pp. 874–880.

Stranders, R., Farinelli, A., Rogers, A. & Jennings, N. R. (2009*a*), Decentralised coordination of continuously valued control parameters using the max-sum algorithm, *in* 'Proceedings of the Eighth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', Budapest, Hungary, pp. 601–608.

Stranders, R., Farinelli, A., Rogers, A. & Jennings, N. R. (2009*b*), Decentralised coordination of mobile sensors using the max-sum algorithm, *in* 'Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)', Pasadena, California, USA, pp. 299–304.

Stranders, R., Munoz de Cote, E., Rogers, A. & Jennings, N. R. (2010), 'Non-myopic bounded approximation for infinite horizon patrolling with mobile sensors', *Artificial Intelligence Journal (AIJ)* . In preparation.

Stranders, R., Rogers, A. & Jennings, N. R. (2008), A decentralized, on-line coordination mechanism for monitoring spatial phenomena with mobile sensors, *in* 'Proceedings of the Second International Workshop on Agent Technology for Sensor Networks (ATSN)', Estoril, Portugal, pp. 9–15.

Stranders, R., Rogers, A. & Jennings, N. R. (2010), A decentralised coordination algorithm for maximising sensor coverage in large sensor networks, *in* 'Proceedings of the Ninth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', Toronto, Canada, pp. 1165–1172.

Thomassen, C. (1994), 'Grötzsch's 3-color theorem and its counterparts for the torus and the projective plane', *Journal of Combinatorial Theory, Series B* **62**(2), 268–279.

United States Air Force (2009), United states air force unmanned aircraft systems flight plan 2009–2047, Technical report, Headquarters, United States Air Force, Washington DC.

Vargas, J. E., Tvalarparti, K. & Wu, Z. (2003), Target tracking with bayesian estimation, *in* V. Lesser, C. L. Ortiz, Jr. & M. Tambe, eds, 'Distributed Sensor Networks', Kluwer Academic Publishers, chapter 5, pp. 219–256.

Vidal, R., Rashid, S., Sharp, C., Jin, S. & Sastry, S. (2001), Pursuit-evasion games with unmanned ground and aerial vehicles, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)', pp. 2948–2955.

Voice, T. D., Stranders, R., Rogers, A. & Jennings, N. R. (2010), A hybrid continuous max-sum algorithm for decentralised coordination, *in* 'Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI)', pp. 61–66.

Waldock, A., Nicholson, D. & Rogers, A. (2008), Cooperative control using the max-sum algorithm, *in* 'Proceedings of the Second International Workshop on Agent Technology for Sensor Networks', pp. 65–70.

Wark, T., Corke, P., Sikka, P., Klingbeil, L., Guo, Y., Crossman, C., Valencia, P., Swain, D. & Bishop-Hurley, G. (2007), 'Transforming agriculture through pervasive wireless sensor networks', *IEEE Pervasive Computing* **6**(2), 50–57.

Warneke, B., Last, M., Liebowitz, B. & Pister, K. S. J. (2001), 'Smart dust: Communicating with a cubic-millimeter computer', *Computer* **34**(1), 44–51.

Weiss, Y. & Freeman, W. T. (2001), 'On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs', *IEEE Transactions on Information Theory* **47**(2), 723–735.

Wooldridge, M. & Jennings, N. R. (1995), 'Intelligent agents: Theory and practice', *Knowledge Engineering Review* **10**(2), 115–152.

Zhang, B. & Sukhatme, G. S. (2007), Adaptive sampling for estimating a scalar field using a robotic boat and a sensor network, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)', Roma, Italy, pp. 3673–3680.

Zhang, T., Madhani, S. & Van den Berg, E. (2005), Sensors on patrol (SOP): using mobile sensors to detect potential airborne nuclear, biological, and chemical attacks, *in* 'Proceedings of the IEEE Military Communications Conference (MILCOM)', Vol. 5, Atlantic City, New Jersey, USA, pp. 2924–2929.

Zhang, Z. (2004), Investigation of wireless sensor networks for precision agriculture, *in* 'Proceedings of the ASAE Annual International Meeting 2004', Ottawa, Ontario, Canada, pp. 1157–1164.

Zhou, J., De Roure, D. & Vivekanandan, S. (2006), Adaptive sampling and routing in a floodplain monitoring sensor network, *in* 'Proceedings of the Second IEEE International Conference on Wireless and Mobile Computing Networking and Communications (WiCOM)', Montreal, Quebec, Canada, pp. 85–93.