Agent-based self-organisation for task allocation reinforcement learning for emergent multi-agent systems

Creech, Niall

Awarding institution:
King's College London

Department of Informatics

King's College London

Agent-based self-organisation for task allocation

REINFORCEMENT LEARNING FOR EMERGENT MULTI-AGENT SYSTEMS

Author
Niall Creech

Abstract

There are many systems where tasks must be allocated amongst multiple, distributed agents, and where each participant must manage its limited resources to best complete these tasks. In stable environments with low numbers of agents there are algorithms to search for the best task and resource allocations. In these types of systems strategies can be planned, and agents coordinated, in a centralised manner.

In more complex situations, such as where there are large numbers of agents, or the environment is highly dynamic or uncertain, these types of solutions do not perform as well. Many real-world systems however are both complex and subject to environmental perturbations, e.g. wireless sensor networks, the coordination of vehicles in smart cities, and the orchestration of drone swarms. In this thesis, we provide contributions towards the challenges of task and resource allocation in dynamic multi-agent systems. We develop decentralised algorithms that are scalable, that work with an agent's local knowledge to improve task and resource allocations in order to optimise the utility of a system in precisely these kind of realistic scenarios.

We develop three contributions to cumulatively solve these problems. As a first step, we develop a reinforcement learning based algorithm to optimise the allocation of tasks based on their quality of completion by agents, while adapting the algorithm in response to an agent's judgement of its historical performance. We next develop an algorithm that allows an agent to allocate its limited resources in such a way as to optimise its performance on completing tasks it has been assigned by other agents, learning the value of these tasks to those agents through reinforcement learning. For our final contribution, we combine these algorithms to provide a holistic solution to the problem of task and resource allocation in dynamic environments, while also extending it to make it more robust to environmental perturbations such as communication disruptions and harsh weather conditions.

We evaluate these contributions individually, through the simulation of different representative systems, before evaluating our holistic solution through a realistic case study in an ocean-based environmental monitoring system.

Acknowledgments

My thanks go out to my supervisors, Dr. Natalia Criado, and Dr. Simon Miles. Through their constant guidance and advice, they have helped me greatly improve my understanding of the field over the course of a number of years. In doing so they have dedicated large amounts of their time to help me develop my ideas, review my writing, and formalise my explanations, all of which I'm truly appreciative of.

For their endless supply of patience with my work-life balance, help to keep me going when my morale was low, and generally putting up with me when I was struggling, I am forever grateful to my family; my wife Sarah, and my children Charlie and Emily.

Lastly I'd like to thank my dad Angus, who was so proud and encouraging of me when I started my PhD, but unfortunately never got to see it finish.

Contents

Chapter 1

Introduction

This chapter introduces the work that will be covered in this thesis. We will highlight the challenges that motivate this research and the applications that make this area of study highly relevant. We will also describe our research objectives, and the layout of the thesis as a whole.

1.1 Overview

What are multi-agent systems?

There are many situations where different *agents*, entities that act autonomously, are attempting to achieve their own goals, a shared group goal, or an overarching system goal. It is often beneficial for agents' actions to be coordinated, so that they can behave cooperatively, e.g. a group of autonomous vehicles organising amongst themselves to avoid collisions or congestion. Whether we are talking about robots, drones, or human societies, these can all be viewed as *multi-agent systems (MAS)*.

Complexity in the real world

Existing in a real-world environment adds complexity to these systems. For example, a fleet of drones trying to organise themselves to fly in formation may experience communication noise due to interference, or wind pushing them off course (See Figure 1.1). The failure of a drone can also impact how others in the fleet must plan their actions. Component wear might cause actions to become *non-deterministic*, i.e. the outcome of a drone taking an action becoming less predictable, or even the complete failure of a drone in the system. To design behavioural rules to handle all these possible states and outcomes over the lifetime of a system would be exceptionally complex and inflexible. Instead, we can design the agents to be intelligent, to learn from experience and to proactively adapt their actions to optimise their behaviour as the system changes.

The task allocation problem

In many cases, there is a set of tasks and subtasks that a group of agents needs to complete to achieve their goal. For example, a group of mobile robots are deployed in a harsh and remote environment, then given the task of collecting soil samples across a wide, distributed area. How then are the individual measurements or subtasks shared out between the agents so that the overall task is completed most efficiently? What if the robots are already spread out geographically, have different functionality, or power supplies? This is the *task allocation problem* for multi-agent systems, how a set of tasks can be best allocated amongst agents in a system to complete.

The resource allocation problem

When completing tasks or taking other actions, agents often need to utilise some resources. When a software agent handles a task it requires CPU time and memory storage. A mobile robot taking measurements may need to use battery power to move around (See Figure 1.2). If an individual agent has multiple tasks to complete, but limited resources to do so, how does it manage the use of these resources to complete the tasks with the best overall outcome? This *resource allocation problem* is common in numerous real-world systems and represents another opportunity for agents to learn optimisations dynamically.

Organising multiple agents

Given a group of agents, an environment for them to operate in, and some tasks to complete, there are numerous ways they could be organised. One could act as a leader, ordering the others to take actions. They could all act in isolation, being purely selfish in their choices. They could form clusters, where small groups of agents act together, perhaps with a local leader, and communicate with other clusters to complete the tasks. Different approaches each have their own strengths and weaknesses, however, in realistic scenarios the system needs to be robust in the face of failures. Agents must be able to adapt to the complex and dynamic environment they operate in. This means that some flexibility in the organisational structure of agents is required.

One way to do this by enabling agents to self-organise, to change the network of other agents they closely interact with, and to alter this throughout their lifetimes to best handle changes in the system. For example, a large expanse of ocean monitoring sensors could be formed into clusters to take measurements, aggregate the results, and then broadcast them back to a base station inland. However, over time they may move with currents, degrade, or fail due to the harsh environment. If they were self-organising, they could reorganise themselves into different groupings to adapt their structure in an attempt to continue to operate effectively.

Real-world problems

Recently years have seen growth in the research and commercial applications of multi-agent systems[1], [2]. Advancements in artificial intelligence techniques have enabled agents to have greater learning capability. Processors, batteries, and other components have become smaller, more powerful, and cheaper, so individual agents have increased resources to plan their actions and adapt their behaviour. Across numerous industries automation and robotics are increasingly commercially viable. For these reasons, research into algorithms that can be applied to agents in distributed multi-agent systems has grown in importance and practical uses.

Vehicle-to-everything (V2X)

For instance, over the last few years much research has been dedicated to the development of autonomous vehicles, with a number of well-funded companies vying for the commercial edge. As part of this, solving vehicle-to-everything (V2X) problems, how each autonomous vehicle communicates and plans tasks as part of a group of mobile agents[3]–[5], has become more prominent.

Figure 1.1: Drone usage and applications. *Drones have become increasingly affordable, light, and powerful. Getting large swarms of drones to operate together collaboratively can be orchestrated through computer sequencing. However, use cases such as long distance deliveries, or monitoring large expanses of agricultural land require much more autonomy.* *(source: nasa.gov)*

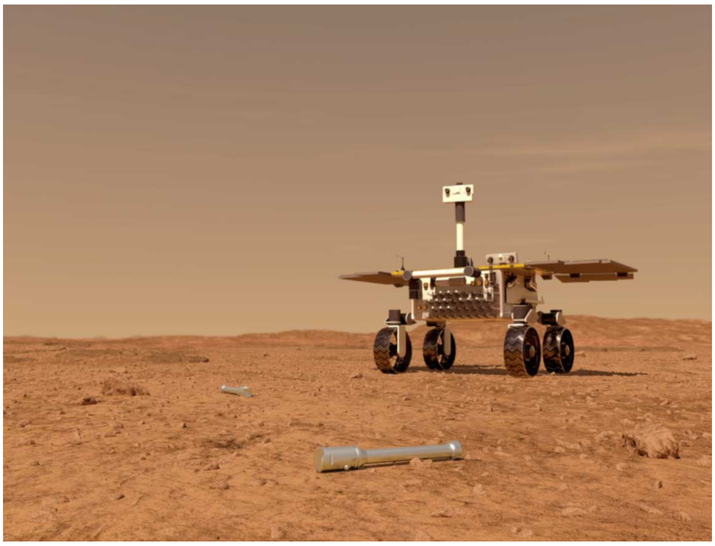


Figure 1.2: NASA Mars rover. *Nasa's Mars rover has many tasks to perform; moving around the planet's surface, sampling rocks, atmospheric experiments, etc, with limited power availability and working lifespan. The rover can optimise performance, minimise wear, and extend its working life, through intelligently allocating its limited resources amongst its many functions.* *(source: nasa.gov)*

Unmanned autonomous vehicles (UAV)

With drones becoming cheaper and more readily available, there are new uses being found for swarms of these unmanned autonomous vehicles (UAV) to cooperate in areas from agriculture[6] to search-and-rescue situations[7]. Climate change research, and the impact of changes to ocean temperatures and conditions on the natural world, has increased focus on monitoring and data gathering to understand these often remote and harsh environments. There is a need for robustness to handle tough conditions, and intelligence to adapt to dynamic natural environments. As a result, the quantity of multi-agent systems research has grown, as well as specifically across the wireless sensor networks (WSN) field as a whole[8]–[11].

We look in more detail at WSN, and their applications, in Chapter 7, some of which we use as explanatory examples throughout the thesis, including some more specific cases such as vehicle-to-everything (V2X) systems, and ocean-based wireless sensor networks.

Why is this research important?

Tackling the challenges presented by the complexity of real-world multi-agent systems give us the goals of our research:

- Adapting to complex and dynamic environments with many interacting agents.
- Planning task allocations amongst distributed agents.
- How each agent manages its resources to complete tasks.
- How agents self-organise, adopting roles that allow them to effectively complete tasks.

The rapid growth and spread of real-world applications motivates this work. We believe that successful solutions to the problems we cover in this work have broad, cutting-edge applicability across many industries and research areas.

Research highlights

Over the course of our work we will demonstrate new algorithms designed for complex, realistic, multi-agent systems. We show how our algorithms optimise for task and resource allocation in disrupted environments, tackling the various challenges through;

- dynamic system exploration by policy adaptation using an agent's historical data to predict how well its performance is in context of the whole system (the RT-ARP algorithm, see Chapter 9, Section 9.4.5);
- enhanced recovery of performance after system perturbations by retaining information learnt about the environment (the SAS-KR algorithm, see Chapter 9, Section 9.4.2);
- optimisation of resource allocation through learning and balancing the goals of other agents (the MG-RAO algorithm, see Chapter 10, Section 10.3);
- the emergence of roles and self-organisation resulting from implicit coordination between agents to complete tasks (see Chapter 11, Section 11.2.3).

Where other algorithms can be seen to perform well in many systems (see Chapter 5, Section 5.6, Table 5.3), our work focuses on the type of dynamic, and perturbed systems common to applications such as wireless sensor networks, where current state-of-the-art algorithms are difficult to apply due to the need of a holistic solution to the multiple problems found in such systems to enable effective performance (see Chapter 9, Section 9.6, 'Methods of analysis'). We will benchmark our work using the utility of our algorithms as compared to theoretical baselines such as the maximum utility possible within a simulated system (see Chapter 9, Section 9.6, ’Theoretical system optimal utility as a baseline comparison’), alongside sensitivity analysis approaches such as comparing how efficiently our algorithm explores the environment given different initial conditions (see Chapter 9, Section 9.7).

1.2 Research Goal

> The goal of this thesis is to optimise the aggregate quality of a stream of incoming tasks through decisions on the allocation of tasks to agents within a dynamic population and allocation of agents’ resources between tasks.

We assume tasks requiring completion to be atomic, in that they can be completed by individual agents without collaborating with other agents, or needing to be executed in a particular order.

1.3 Challenges

In the course of our work we focus on reinforcement learning techniques to enable agents to learn strategies to complete tasks to best achieve system goals, where these approaches are not restricted in applicability due to limitations on their computational scalability. We see how learning algorithms can generate self-organisation amongst agents, where they act autonomously and learn to assume differing roles. This leads to a number of challenges;

1. A system may contain a set of agents capable of completing different tasks, to different degrees of quality. How do agents that have been allocated a set of tasks proactively learn which group of other agents can best help them complete those tasks in order to achieve the highest aggregate quality? The introduction of computational constraints means agents must also balance collecting significant knowledge about a small subset of agents in the system or less about a larger set of agents;

   Choosing agents to cooperate with

2. An agent that has multiple tasks allocated to it, but finite resources available to complete them, must prioritise its actions by allocating more resources to those of its actions that are most useful to the achieving the goals of the system. This may be through the prioritisation and completion of tasks itself, or co-ordination with others to have those agents complete tasks. How can an agent learn to adapt its resource allocation strategy to best predict what tasks it may be asked to complete, and the value of its role in doing so?

   Allocating limited resources

3. An individual agent can learn to provide functionality to other agents and form complementary roles within a group that helps meet the overall goals of the system. When an agent places a higher probability that it will take a certain subset of available actions that proves valuable, it assumes the role defined by that set of actions.

   Self-organisation and roles

1.4 Objectives

We form 3 objectives for our research that will allow us to meet our research goal.

Optimising task allocation

Objective 1. Learning the optimal allocation of tasks in a multi-agent system under resource constraints
A system contains a set of agents, and a set of tasks, which may be composed of multiple subtasks. Each agent can take actions to execute or allocate subtasks to other agents. How can they learn to distribute and complete the subtasks to produce the optimal system utility?

Where we have a set of agents each with differing efficiency in performing a set of actions we investigate how an agent can choose a group of other agents that help it to achieve its goals. The locality of the other agents, and size of the group chosen to collaborate with, is restricted by the agent's resource constraints. These fix the amount of knowledge that can be collected, stored and processed by an individual agent. For example, in real-world systems these constraints may be due to CPU, memory and storage limits of the hardware involved. While an agent will attempt to learn its optimal group of agents to communicate with, those agents will also be dynamically learning their own groups and behaviours, and optimising for their own goals.

Optimising resource allocation

Objective 2. Learning the optimal allocation of agents' resources to complete tasks
If an agent continuously receives sets of tasks allocated to it by set of agents, how can it learn to optimise its allocation of its limited resources amongst each type of task it receives? In order to maximise the system utility, the agent must learn the value of the tasks it performs to the allocating agents, and optimise for that value. Additionally, if the allocating agents are able to allocate tasks amongst multiple agents then the value of a task's completion may vary depending on the distribution of tasks amongst agents.

Assuming roles

Objective 3. Self-organisation and the emergence of roles to enhance cooperation and planning
Agents in a system can take a range of actions, executing tasks, allocating them to other agents, or learning about other agents in the system. Given a set of tasks, can agents self-organise and assume different roles that help to optimise the system's utility? For instance, an agent could focus on improving its execution of tasks, optimise its communications with other agents for the better re-allocation of tasks, or on information collection that will allow it to improve the overall planning coordination amongst agents.

Many tasks in a system require some level of collaboration to achieve an overall goal. An agent that is cooperating with another agent can learn how to carry out actions that help the other agent attain its goals. In doing so, the agent can receive rewards depending on the value of its behaviour to the agent it is assisting. In this way, an agent can learn to adapt its behaviour and adopt a distinctive role within the system.

Figure 1.3 shows how an agent might optimise its performance by learning the best agents to carry out a set of actions. Figure 1.4 shows a pair of agents learning to

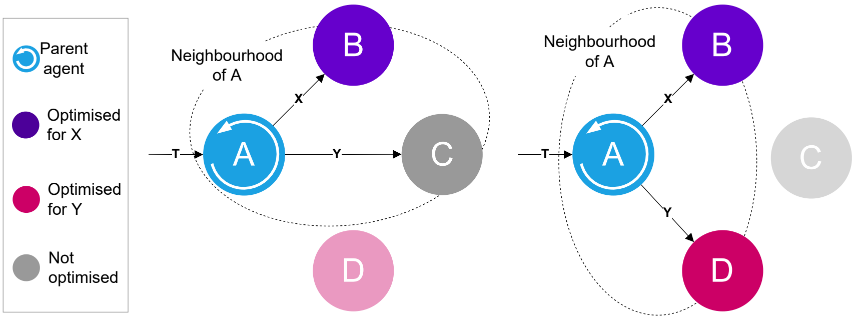


Figure 1.3: Task allocation in an agent system. *Agent A repeatedly receives sets of tasks of type T, which requires it to carry out subtasks of type X and Y. In the first diagram it requests the most optimal agent, B to carry out tasks of type X, but requests Y from agent C, which is not most optimal agent for these subtask types. In the second diagram, agent A has learned that agent D is the best choice for tasks of type Y and has altered its neighbourhood to exclude C and include D instead.*

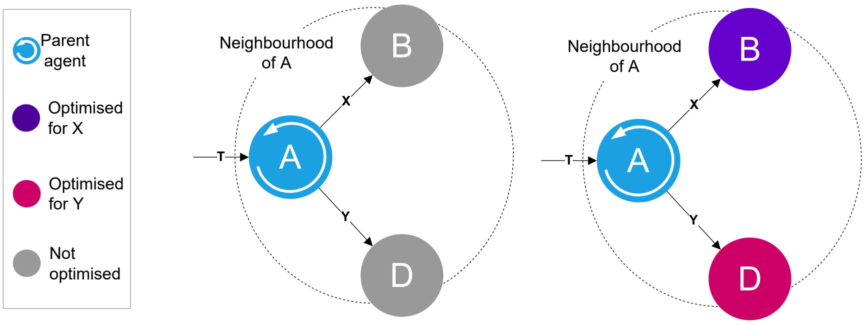


Figure 1.4: Optimising actions in a multi-agent system. *The two figures show a pair of agents learning to optimise their actions in order to respond to incoming requests. In the first diagram agent A allocates subtasks of type X to agent B, and Y to agent D. Neither agent has optimised its resources to prioritise either of the possible tasks. In the second diagram, agents B and D have reallocated their resources to optimise the value of them completing tasks X and Y respectively, to agent A.*

optimise their resource allocation to maximise the value of task completion to another agent that allocated them the tasks. In Figure 1.5 we see how agents can develop roles in the completion of tasks.

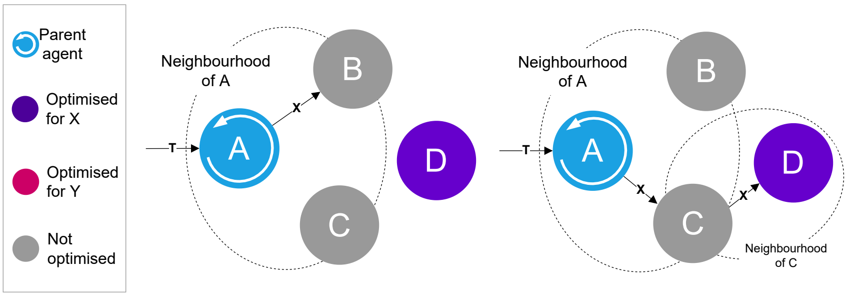


Figure 1.5: Neighbourhoods in a multi-agent system. *In the first figure, agent A is allocating tasks of type X to agent B, which is not optimised to perform these tasks. An agent D is optimised for these tasks but is not within the neighbourhood of A and cannot be reached by it directly. Agent C is in the neighbourhood of A, but again, is not optimised to perform tasks of type X, however, it can reach agent D. In the second figure, agent C has learned to adopt the role of relaying tasks for A, and as a result, A has indirectly extended its neighbourhood to reach agent D and have tasks X optimally performed.*

1.5 Contributions

To create an overall algorithm to tackle our research goal we first study the objectives mentioned separately, then combine the solutions we develop for each, to form a holistic solution. Given this, we detail the three contributions of our research below.

Contribution 1. Task allocation algorithms
We present four algorithms which, in combination, enable each agent to improve their task allocation strategy through reinforcement learning, while changing how much they explore the system in response to how optimal they believe their current strategy is, given their experience. These algorithms allow an agent to determine the capability of other known agents to perform tasks, allocate these tasks, and carry out other actions based on its current knowledge and the need to explore agent capability space.

Contribution 2. Resource allocation algorithms
We introduce an algorithm to optimise resource allocation which uses multiple function approximations of the demand on agents' resources over time, alongside reinforcement learning techniques. This method is applicable where there are competing demands for shared resources, or in task prioritisation problems.

Contribution 3. A combined algorithm for hierarchical multi-objective task and resource allocation
To adapt to the changing agent composition in a dynamic system, and the associated change in agent capabilities available, we develop an algorithm that utilises and extends the first two contributions. The algorithm solves for the challenge of optimising for multiple, competing objectives, while enabling agents to learn to adopt different roles within the system.

With these three contributions we tackle the objectives set out in Section 1.4 that arise from the challenges we described in Section 1.3. In doing so, we develop a solution that is capable of meeting our research goal as defined in Section 1.2.

1.6 Outline

The rest of the thesis is broadly laid out into three parts. In Part I we introduce the relevant concepts, theory, and existing work on multi-agent systems. Part II covers our main research work, the algorithms we contribute, and their evaluation in a realistic case study. Finally, Part III examines our work, reviews what has been achieved in the thesis, and looks towards future work and applications.

Part I - Background

In Chapter 2 we introduce the high-level ideas and concepts around multi-agent systems, the environments they operate in, and how they apply to distributed systems. The subsequent chapters go into detail on relevant areas such as; the problems involved in task allocation in Chapter 3, the prioritised allocation of limited resources for completing tasks in Chapter 4, and the use of reinforcement learning in multi-agent systems in Chapter 5. In particular, we look at how such reinforcement learning algorithms can be used to optimise large, distributed systems, and the problems that can result when they are applied to real-world environments. In Chapter 6 we examine the organisational structure of distributed agent systems, their defining characteristics and self-organisational behaviours. Finally, Chapter 7 focuses on wireless sensor networks, examples of which we use to illustrate concepts throughout our work, with an ocean monitoring example forming our case study in Part II.

Part II - Research contributions

Chapter 8 formally sets our agent-based system, and where we introduce the notation and concepts that define the task and resource allocation problems that we will subsequently look to solve. To build a full solution we tackle the problem in three distinct blocks. In Chapter 9 we focus purely on algorithms to tackle the task allocation challenge. We then develop algorithms that target the allocation of agents' resources to prioritise tasks in Chapter 10. This leads into Chapter 11 where both the task and resource allocation solutions are brought together to solve the overall problem stated in Section 1.2. In doing so, the solution is naturally extended to enable increased coordination between agents, allowing self-organisational structure to develop within the system. In the last section of work in Chapter 12, we develop a case study based on an oceanographic environmental sensor network. Agents in this system operate in a harsh and dynamic environment where they require autonomy in order to continue to function reliably. This allows us to evaluate our algorithms in a realistic scenario and judge their performance.

Part III - Applicability and analysis

Finally, in Part III, Chapter 13 we summarise what our research has achieved, and how well it met our research goals. We also look at some of the possible applications of our work, and where it could be extended or enhanced through future research directions.

Part I

Background

Chapter 2

Introduction to distributed multi-agent systems

This chapter introduces the high-level concepts and definitions that apply to multi-agent systems in general, and to distributed task allocation problems more specifically. We will talk in detail about the environments agents operate in, and how they can be organised. This allows us to define the key characteristics of the systems we will work with in future chapters and why those characteristics are essential to real-world applications.

2.1 Introduction

In distributed *multi-agent systems (MAS)* there are interactions between many independent actors. These systems are seen in a wide range of real world applications such as wireless sensor networks (WSN)[1] [11]–[14], robotics[15], [16], and distributed computing[17], [18]. The growing complexity and scope of these applications presents a number of challenges; responding to change, handling failures, and optimisation of agents' actions. The performance of the system must also be scalable with growth in the number of agents, and be able to perform tasks given constraints on computational, storage, or other resources.

The challenges summarised below are shared across many diverse subject areas, so solutions to them are applicable across a broad range of fields;

- *task allocation*, how can tasks be allocated amongst agents in a system to achieve the best results? Note, an agent may have a goal that consists of an overarching task that requires the completion of a number of subtasks by other agents[19] (Chapter 3);
- *ad-hoc dynamic networking*, how adaptable is agent discovery and communication? Agents must be able to communicate with each other while connections are lost and created[20] (Chapter 3);

[1] WSN systems are covered in detail in Chapter 7.

- *resource allocation and management*, how are resources best allocated by agents to complete tasks? E.g. managing energy usage while performing a function within a physical environment[21]–[23] (Chapter 4);
- *self-organisation*, how do agents autonomously adapt their organisational structures to help them complete goals? Solutions with fixed architectures are often non-applicable to dynamic systems with many unknowns as designs would be too complex and inflexible. To improve agents' adaptability in these situations, learning algorithms can be used[24]–[28] (Chapters 5 and 6).

Chapter structure In the next sections we define multi-agent systems at a high-level in Section 2.2, how they can be designed in Section 2.3, categorise the types of environments they operate in Section 2.4, and the challenges they face in Section 2.5.

2.2 Defining a multi-agent system

The systems we consider contain agents that are attempting to complete a range of tasks, while operating within a dynamic environment. This makes them closely identified with concepts such as *autonomy*, *locality of view*, and *decentralised planning*. Agents are autonomous and have some degree of freedom to make their own decisions. Locality of view arises as agents do not have a full, over-arching knowledge of the system that they can use to make decisions, only partial information. As the system does not rely on centralised control or orchestration, there is decentralisation of planning, and of organisational structure.

Systems with these properties fall naturally into the study of multi-agent systems. We can define an *agent* as a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives[29]. With this definition, *multi-agent systems (MAS)* are systems composed of autonomous agents, with distributed control, data, and computation.

Intelligent agents Intelligence is a characteristic of agents that we need in complex systems in order to generate adaptive autonomous behaviour without intractable amounts of initial external design. We define these properties as[2];

1. *proactiveness*, intelligent agents are able to exhibit goal-directed behaviour by taking the initiative in order to satisfy their delegated objectives;
2. *reactivity*, intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their delegated objectives;
3. *social ability*, intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

The need for proactivity and reactivity is inherent due to the uncertainty in a MAS arising from the dynamism caused by the interactions of multiple agents. Agents cannot be sure of the outcomes of their actions or that the environment has not changed in a way they are not aware of yet. In these circumstances they must be able to react to unexpected outcomes, and investigate ways to adapt their actions and strategies to compensate for them.

[2] See Wooldridge and Ciancarini[29]

Agent coordination

With many agents in an environment simultaneously attempting to carry out tasks, there is a benefit in them being able to coordinate amongst themselves. This may be simply as they are altering the same environment, so planning actions with knowledge of each other's effects on the environment could produce better outcomes for each agent. E.g. a group of mobile robots taking samples of soil across an area of land could avoid duplication or collisions by communicating their intended actions, or the outcomes of previous ones, amongst themselves.

For some tasks or goals the agents will have to actively orchestrate their behaviours to achieve a mutually beneficial outcome. For a complex task composed of many subtasks there could be a required sequence to achieve the goal, such as when a robot must utilise a range of motors with the right order and timing to maintain its balance while moving a leg.

2.3 Designing multi-agent systems

There are numerous ways to design MAS to have intelligent behaviours. We describe some of the main ones below, and the reasoning behind our choice of approach.

Logic-based strategy

One way is to use a *logic-based* approach, symbolically representing the state of an environment, working with this representation to logically solve the problem specified, and then setting out the behaviours required by the agent to reach the desired state. This method may also be used in combination with modern machine learning techniques[30].

Reactive architectures

Reactive architectures are based on the concept of environmental sensing and responding to actions based on pre-encoded knowledge contained within the agent, where expert knowledge can be distilled into rules that can be activated when the system is in certain states. These rules can be layered as in subsumption architectures[31] to form hierarchical behaviours (e.g. decomposing the walking motion of a robot into individual joint movements). In large dynamic systems this task of encoding rules and logic quickly grows incredibly complex and difficult to design[32], [33].

Belief-desire-intention

The *belief-desire-intention (BDI) model*[34] separates out an agent's actions from its planning. The belief represents the agent's knowledge of the system, which may or may not be true. Desires are things the agent would like to achieve, which can then generate goals. Intentions are things the agent has chosen to do to achieve its goals, with these leading to the choosing of actual plans to carry out. This design concept can be extended to include *obligations*, social norms within an agent-based system[35].

Self-organisation and emergence

We can also approach MAS design using *self-organisation*[36], [37]. In this case, the organisation and reorganisation of the system happens without any explicit commands being given from an external source. Linked to this in multi-agent systems is the idea of *emergence*, where the higher-level behaviour and properties of the system are the result of local, low-level agent behaviours.

The complexity and distributed nature of many of the applications of multi-agent systems makes pre-designing systems intractable in many cases, which is where self-organising approaches become attractive for solving task and resource allocation problems in large, dynamic systems. We will discuss this area in-depth in Chapter 6 as the basis for understanding self-organisation and emergence in our work on agent roles in Chapters 11 and 12.

2.4 Environments

Agents will exist in some *environment* in which they must operate. There are a number of key concepts that will be helpful in discussing the environment of MAS systems[29], [38], [39], which we detail below.

Global view To make decisions, agents must have a view of their surroundings. Through sensing the state of the system they can choose actions and observe changes as the system evolves. If agents could measure all the information about the state of the system relevant to their action choices, the system is considered *fully observable*, where the agents have a *global view* of the system's state. E.g. agents controlling chess pieces on a chess board are likely to have full knowledge of the state of all the pieces on the board before acting.

Partial view In many cases agents may only be able to have a subset of knowledge about the system they are operating in. If the system state is very large, then an agent might not have the computational resources or memory to observe, and process, the full state. Its ability to sense the system may be limited, such as when a robot's visual detectors are blocked from seeing further by objects in the environment. There also may be noise resulting from sensors having limited accuracy or if they develop faults. This means that agents must make decisions based on incomplete information, using their *partial view* of the system state (See Figure 2.1).

Discrete and continuous states Where there is a finite set of states a system can be in, it can be said to be *discrete*. Where there is no such bound it is *continuous*. If a set of agents were deployed in an environmental setting there would be an infinite number of locations, and therefore states, that each agent could be in. In practice these states can often be quantised. In the case of locations, a granular location grid is often used instead of exact coordinates, which reduces the number of states to a finite amount (See Figure 2.2). Note that even when the set of states is finite, such as with a chess board, the number of possible states can be so large as to be viewed as continuous from an agents perspective in terms of computational demands and complexity.

Cooperation and Coordination Agents working cooperatively often need to communicate their intentions, actions, and results of those actions to other agents to enable them to work together and avoid conflicting behaviours. E.g. autonomous vehicles can communicate their planned actions to other vehicles so that collisions and congestion can be avoided. There are different ways agents can plan their actions together. *Centralised coordination* means that one agent or controller will collect information about all the other agents in the system, construct the optimal plan for their combined actions, then communicate these back to the agents.

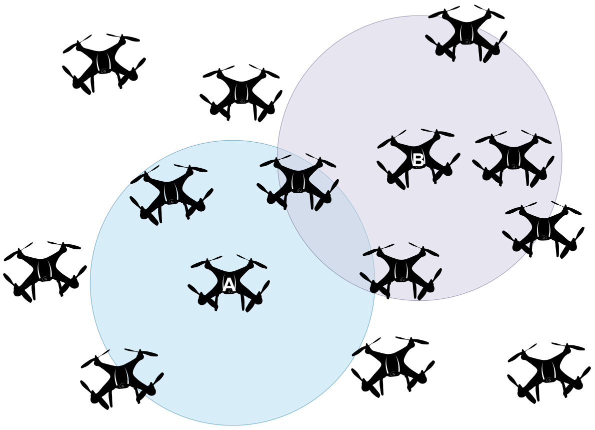


Figure 2.1: A fleet of UAVs flies in formation. *Drone A can detect the other drones immediately around it, its partial view of the system. Drone B has a different partial view of the system, overlapping that of drone A. The agents will use these partial views to plan their actions. As each agent's partial views of the system may overlap, the overall, partial global plan will approximate the optimal global plan for coordination of all the drones movements, i.e. the plan that would be constructed if the system were fully observable.*

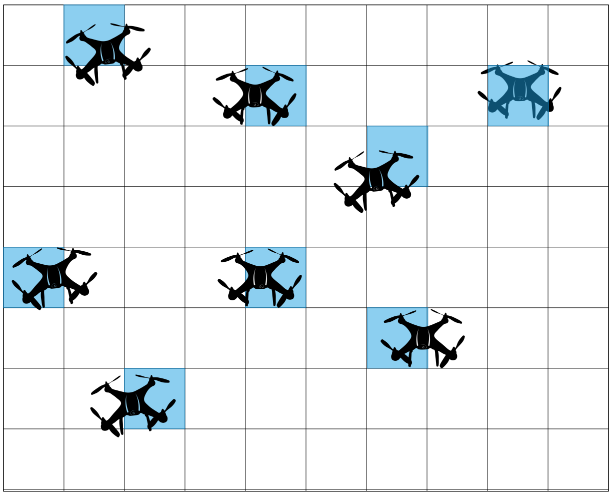


Figure 2.2: Discretization of UAV locations. *Each drone in a fleet of UAVs can have its position described by a set of real numbered coordinates. This means that there is an infinite set of states that each can be in, and the system is continuous. By quantising the map of the deployment area of the drones, the set of locations becomes finite, and the system is discrete. With the reduction in states, the demand on computational resources to calculate planning algorithms is decreased, at the risk of a reduction in their accuracy.*

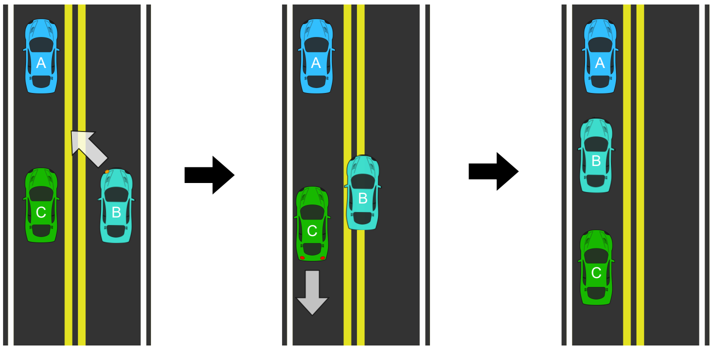


Figure 2.3: Coordination in a V2X multi-agent system. *In the first figure, an autonomous vehicle, B, independently plans to move into the left-hand lane, where there are currently two other vehicles, A and C. Through exchanging information on their plans with each other, each vehicle adapts its plan in response to the new knowledge they now have about each others intentions. As a result, C changes its plan to slow down and make space for vehicle B. This allows for decentralised planning, but with explicit coordination to reduce the chances of collision between the vehicles.*

Explicit and implicit coordination With *explicit coordination*, agents will develop their plans independently, which are set out to achieve their own individual goals. Each agent then adapts their plan through the exchange of knowledge with other agents in the system (See Figure 2.3). In this way a partial global plan is constructed. The plan is an approximation of the optimal global plan, such as that created through centralised methods, as it is based on the combination of the multiple, simpler plans that each agent has created using only their partial information about the system.

An agent may learn information through other means that don't involve direct communications with other agents. This *implicit coordination* through environmental signalling is the result of agents all interacting with the same environment. E.g. when an ant leaves a pheromones trace in the environment, this acts as a method of coordination for other ants to adapt their behaviour accordingly[40].

Deterministic actions In a simple system the result of an agent's action may be completely *deterministic*. Given the state of a system, and an action, the subsequent state is guaranteed (See Figure 2.4). E.g. given one state of a chess board, and a one move of a piece, the position of all the chess pieces after the action is completely predictable. If there are multiple agents acting within the system, but the outcome of their combined actions in one state results in another completely predictable state given those actions, then the environment is *strategic* (See Figure 2.5). However, in many systems the outcome of an agent's actions can be uncertain due to the environment it is operating in. For example, an agent controlling a drone, operating in the natural environment, could take identical actions to change direction, when in an identical state, but the outcome could differ due to the effects of the weather blowing it off course. In this case the system is described as *stochastic* (See Figure 2.6).

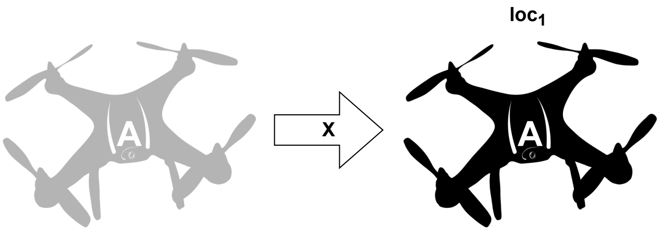


Figure 2.4: Deterministic actions in a drone-based multi-agent system. *An airborne drone, A, takes an action X, in a deterministic environment. The result of the action moves it to the predicted position loc_1 with probability 1.*

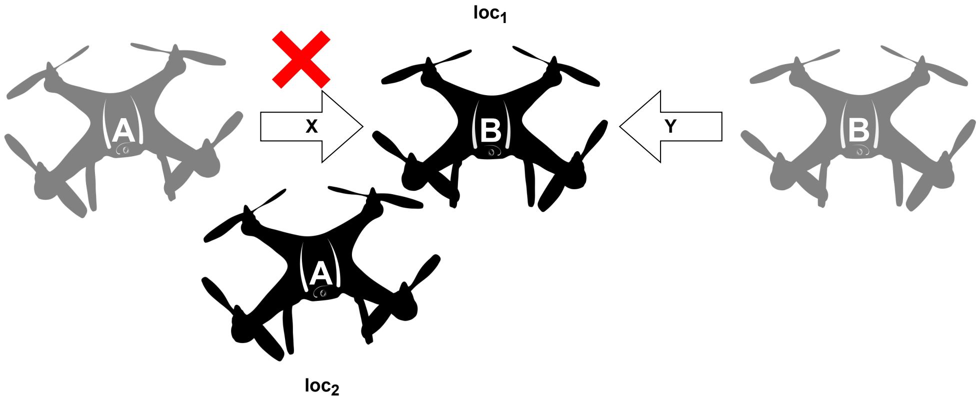


Figure 2.5: Strategic actions in a drone-based multi-agent system. *An airborne drone, A, takes an action X to move to position position loc_1. Simultaneously, another drone B takes an action Y, to also also move to location loc_1, pushing drone A into position loc_2. Although the outcome of drone A's action is not as it planned, if both drones' actions X and Y were known, the outcome would have been predicted. In this case the environment is strategic.*

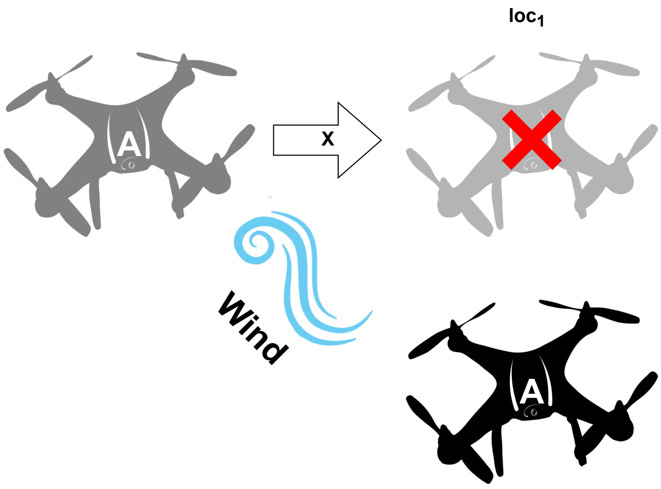


Figure 2.6: Stochastic actions in a drone-based multi-agent system. *An airborne drone, A, takes an action X. The predicted outcome is that it ends up in location loc_1, however, a gust of wind pushes it off course and it ends up in position loc_2. The result of the same action in the same state will sometimes be the desired loc_1, but may randomly be another position. The environment is therefore stochastic.*

Dynamic and static environments When an agent has a view of the state of the system, it then uses this information to calculate its next action, before executing it. If the state of the system is the same between it sensing it, and taking an action, the environment can be said to be *static*. However, if the state changes while an agent makes a decision, perhaps as a result of the actions of other agents, then the system is *dynamic*. Where changes in the environment are caused by external effects such as weather, component failures, communication disruptions etc, we define these as *perturbations* to the system.

Homogeneous and heterogeneous agents An agent will have a set of *capabilities*, the set of actions it can carry out, plus its available computational, memory, and other resources required to perform them. If all the agents in a MAS have the same capabilities, the system can be said to be *homogeneous*. In a *heterogeneous* system however, agents do not have the same capabilities. As a result, the actions available to each agent may be different, as well as their performance in completing them. Note that an environment may start off as homogeneous and become heterogeneous over its lifetime. For example, in a WSN system, nodes might have exactly the same range of sensors available to them when initially deployed, and be homogeneous. However, hardware failures could lead to the system becoming heterogeneous over time, as components degrade and each agent's capabilities diverge.

Sequential and episodic events Sometimes an agent's actions in a state can be considered in isolation, in other words they are not affected by that agent's past behaviour. E.g. the outcome of moving a chess piece in a given state will not be any different if it has been moved in the same way previously. In a *sequential* case however, past performance will affect the outcome. If an agent is given a task to perform that uses some resources for example, its performance in carrying out the same task again may be affected by having less or no resources available to it that are required by the task's execution.

2.5 Challenges

State-action space size

Formally designed agents can perform set tasks given a well-understood system. However, it is often not feasible to design algorithms that can predict the large variety of failures or changes that may occur in large-scale, real-world operating environments. In addition, as the systems become more complex there is an exponential growth in the size of agents' *state-action space*. This space represents the set of combinations of states they can be in, alongside the actions they may take in those states. Knowing this space before deploying the agents is often unrealistic, as is understanding which algorithms will perform optimally.

Central points of failure

Introducing a centralised source of continually updated information on the environment and other agents can increase the knowledge available to an agent about their state-action space, allowing for better optimisation. These *orchestrating agents*, agents that specialise in coordinating other agents in the system, can be found in many distributed software architectures[41]–[44] and robotics[45], [46]. However, in utilising this design, a central point of fragility is created, even if the problem can be partially mitigated through clustering and consensus techniques to increase fault-tolerance. As other agents' interactions and communications are channelled through these centralised agents, congestion and bandwidth saturation problems also grow.

Multi-agent reinforcement learning (MARL)

Distributed agent systems with learning enhancements such as *multi-agent reinforcement learning (MARL)* can provide similar functionality but distributed across agents (See Chapter 5). This removes the focal points for orchestration and mitigates congestion issues while still providing the knowledge sharing and action coordination that allow agents to optimise their actions. With an increasing number of interacting agents however we see an exponential increase in the amount of communications within the system, eventually saturating bandwidth and exhausting computational resources. There is also an *expectation of stability*, that the solution to the agents optimisation remains relatively stable, with a gradual reduction in the need for exploration of state-action space over time. In dynamic systems this often does not hold. MARL techniques also do not take account of the inherent risks involved in taking different types of actions, leading to catastrophic effects in areas such as robotics where some actions may risk severe physical damage, or in financial systems where large losses might be incurred[47]–[50].

Challenges of decentralisation

Solutions often target specific systems and technologies but share commonalities due to the underlying theoretical problem being similar. These can be broadly categorised as being centralised or decentralised. In centralised solutions, the behaviour of agents is coordinated through a shared decision-making component. As the environments become more dynamic and the number of interacting agents expands, the complexity of orchestration and communication increases[51]–[53]. The use of hierarchical structures, such as in *holonic systems*, can increase the applicability of this approach, but limits in scalability still exist due to the same issues[54]–[56].

With decentralised approaches, agents behave with at least some autonomy. They have a local view of the system, not a global one, either due to the system being partially observable to them or too complex for them to use system-wide knowledge effectively. Solutions that utilise system-wide information can optimise well, but do not easily scale as the complexity of the systems increase, with calculations becoming intractable. When only local-knowledge is used, scalability is increased, but optimisation can be difficult due to the use of multiple independent solutions based on agents' local views instead of a combined global solution.

Example 2.5.1 (*Examples of MAS systems*)**.** There are numerous problem areas where scalability challenges limit the effectiveness of centralised techniques for orchestration, or where the poor reliability of communications requires some level of autonomous behaviour. Significant research on these problems has been carried out in areas such as;

- Vehicular ad-hoc networks (VANET) and traffic control[3], [4], [57];
- Unmanned autonomous vehicles (UAV) communication and power management[58];
- Routing and energy distribution in wireless sensor networks[59];
- Resource allocation and service scaling in cloud computing[60];
- Quality of service in content delivery networks[61].

2.6 Summary

In this chapter we covered the basic concepts in multi-agent systems, and the broad elements of a distributed multi-agent system. This helps us understand the challenges of different environments and the trade-offs in design techniques. Our work focuses on systems with decentralised planning, using self-organisation to form emergent roles in preference to more pre-designed strategies. The environments are partially-observable, and dynamic, agents are cooperative and heterogeneous, and actions are sequential with strategic outcomes. We also use an element of implicit coordination through the use of rewards.

Chapter 3

Task allocation

Achieving goals within a multi-agent system often requires cooperation and coordination amongst multiple agents to complete complex tasks. The problem of allocating tasks and subtasks so that agents can complete them successfully and maximise the overall system utility is the subject of this chapter (Contribution 1).

3.1 Introduction

There are many systems containing multiple agents where it is useful for an agent to be able to allocate tasks to other agents in order to aggregate information, or improve the outcomes of the individual tasks. E.g. if there were a system containing multiple vehicles communicating with each other, one vehicle may want to understand the congestion levels in its local area. To do so it could allocate congestion measurement tasks to other agents, then collect together the results. The question arises on which other agents to allocate these types of tasks to get the most useful results, and how to perform reliably in a dynamic environment.

Chapter structure

In the next section, Section 3.2, we cover the task allocation problem. We then look at some examples of this problem in real-world fields and key considerations in solving it in Section 3.3. Section 3.4 examines the differences between centralised and decentralised task allocation. Section 3.5 introduces some of the algorithms used for this problem, in particular multi-agent reinforcement learning, as well as their strengths and limitations. The nature of the exploration strategies used by agents in optimising their actions is covered in Section 3.6, followed by some important concepts particular to algorithms operating in large, continuous systems in Section 3.7.

Example 3.1.1 (*Task allocation in an ocean-based sensor network*). In a sensor network set up to monitor ocean conditions, agents each control sensors on individual buoys distributed across the ocean surface. Tasks in this case are salinity measurements to be taken at locations within this area. Agents are heterogeneous, in that they may not perform tasks equally well, perhaps due to component ageing or damage. The tasks are continuous as requests for measurement are repeated over the lifetime of the system. The system is dynamic, as agents' actions may effect the value of the actions of other agents, and perturbed, in that there are currents and other environmental effects. With limited power supplies, and the need for tasks involving measurements to be relatively recent to be useful, there are natural resource constraints on the system.

3.2 Defining the task allocation problem

Given a set of tasks to execute, an agent can allocate them to other agents in the system to help it complete those tasks. The agent can allocate tasks amongst agents it already knows, learning which of those agents complete each different type of task with the best results. The agent can also learn about other agents in the system, which may help it achieve better performance, at the expense of expending more resources discovering these new agents compared to focusing on task completion alone.

The task allocation problem

Definition 3.2.1 (*The task allocation problem*). The *task allocation problem* is that of optimising the allocation of a set of tasks in a MAS amongst agents to best achieve the objectives of the system, given some set of constraints.

3.3 Objectives of a task allocation solution

Many different industries involve multi-agent systems where there are practical examples of task allocation problems. There are close similarities between these problem types in terms of the algorithms that can be applied. In addition, the similarity of the underlying systems across multiple industries makes research in one area highly relevant to many others. As such, although we use multi-vehicle systems and wireless sensor networks throughout our work as examples, many of the insights transfer to other problem domains and industry sectors.

Task allocation in V2X systems

Modelling and coordinating behaviours of actors in *vehicle-to-everything (V2X)* systems is essential for autonomous driving and interconnected transport management. Information exchange allows for vehicles to be more aware of other vehicles[62], and to interact with road infrastructure systems. This means they can make decisions with increased safety and greater efficiency[63]. A request between vehicles may be a composite set of tasks such as providing position and speed data, traffic congestion information[64], traffic light status, roadworks in progress, and so on.

UAV and MANET

Similar problems arise in UAV and other *mobile ad-hoc networks (MANET)* more generally. There may be multiple vehicle-to-vehicle and vehicle-to-ground communications, and restricted energy availability, which needs to be managed effectively[58]. The interactions of multiple mobile vehicles is an area where there is a large body of research on artificial intelligence applications[3], [4] to develop models and predictive solutions.

Key requirements

To achieve a solution for these task allocation problems in multi-agent systems we need to develop the abilities for agents to;

1. learn to make the best decisions given their current state;
2. adapt how they explore state-space depending on how successful they are in task-allocation currently;
3. make decisions based only on a localised or otherwise partial-view of the system;
4. maintain their resource usage within set limits.

3.4 Centralised and decentralised task allocation

Centralised task allocation

In orchestrating the allocation of tasks we can use centralised planning, or decentralised techniques. With centralisation there is central agent that coordinates the other agents. This agent has communication channels open with all the other agents in the system, and can use these channels to gather information about the capabilities of the agents in the system, solve the problem of finding the best allocation of the tasks amongst those agents, then distribute them accordingly[65]–[68] (See Figure 3.1).

The main drawbacks of this approach are due to the centralisation of functionality and resource usage within a single agent. As each agent must communicate with the central agent, scalability in number of agents in the system is limited by the resources required by the central agent to handle the communication. Additionally, the task allocation problem itself is solved globally by the central agent, so the complexity of the computation possible is limited by the resources available to that specific agent. The reliance on a coordinating agent is also a single point of failure for the system, decreasing the robustness of the approach in environments where agent failure or damage are real considerations[1].

Decentralised task allocation

Decentralised task allocation strategies avoid many of the issues that limit centralised techniques, although they have their own issues to be overcome. In this case, there is no centralised controller orchestrating the other agents. Each agent has a local view of the system, and communicates with other agents in the system to coordinate the allocation of tasks[70], [71] (See Figure 3.2).

As agents communicate with a local group of agents, the resources used for communication are spread out amongst agents. This is also true for computational costs as each agent will solve a smaller task allocation problem rather than the full, global problem. With this distribution of the problem across agents, robustness of the system is increased as the loss of an individual agent is less likely to have a critical impact

[1] The use of clusters of primary controllers can be used to mitigate the robustness risk, however there are limits to this as a solution[69].

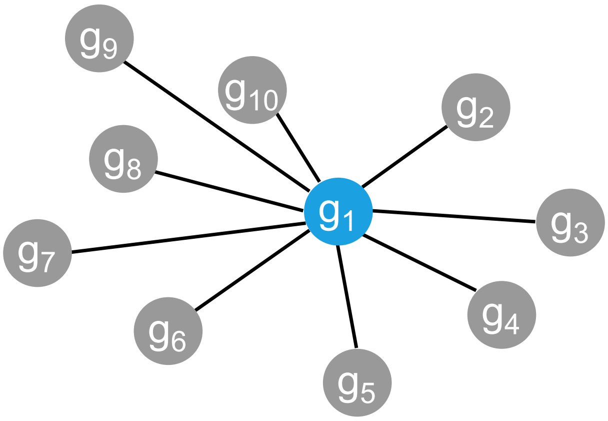


Figure 3.1: Coordination of task allocation in a centralised MAS. *Given a set of tasks to complete, the agent g_1 collects knowledge from all agents g_2 - g_{10} in the system. g_1 then calculates the globally optimal allocation of tasks amongst the agents, before allocating the to the agents. The majority of the communication and computation of the global solution is with agent g_1, which requires ever increasing CPU, memory, and bandwidth, limiting the scalability of this approach.*

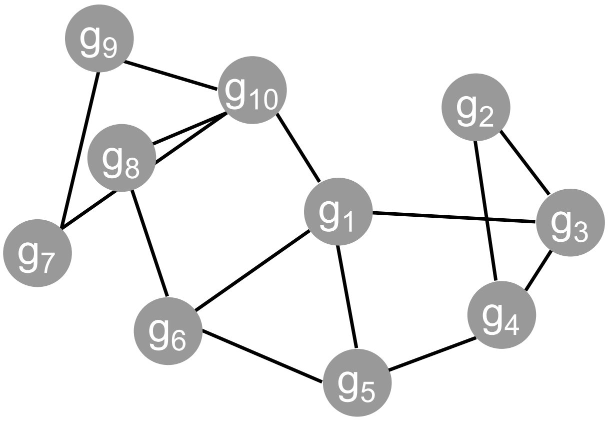


Figure 3.2: Coordination of task allocation in a decentralised MAS. *Agents in the system collect knowledge from a subset of other agents in their locality. Each agent calculates its own locally-optimal view on the allocation of tasks, in coordination with other local agents. The agents then allocate tasks amongst themselves. Communication and computation is now distributed, increasing scalability, at the cost of a more approximate task allocation solution.*

on performance in large systems.

The negatives of this strategy come from the distribution of the task allocation problem itself. When the problem is solved centrally, the knowledge held by all the agents in the system can be used to find the optimal solution. With decentralisation however, as each agent forms a localised view of the system, the task allocation problem is decomposed into many smaller problems that each agent will solve individually. Each agent has both incomplete knowledge of the system as a whole, and of other agents' solutions to the problem. The effect of this is that the overall system solution to the task allocation problem is likely to be suboptimal.

3.5 Multi-agent reinforcement learning algorithms

To provide some context for the work to follow we look at some relevant research in using multi-agent reinforcement learning (MARL) to solve task allocation. Although there are other useful strategies, such as auction-based systems, and particle swarm optimisation techniques, these also have specific challenges. Auction-based systems carry increasing orchestration cost as the number of agents involved increases, which impacts the scalability of related solutions. They also suffer significant impact when the system is dynamic as agent communication is lost. Swarm approaches can be effective under dynamic conditions but are also prone to optimising on local-optima[72].

In discussing MARL, we focus in particular at methods of allocating rewards to drive behaviours, how allocation affects both the exploration of state-space, and coordination between agents.

Multi-agent reinforcement learning

Multi-agent reinforcement learning (MARL)[71], [73], [74] applies reinforcement learning techniques to multiple agents sharing a common environment. Each agent senses the environment and takes actions that cause a transition of the system from one state to a new state, resulting in feedback being given to the agent in the form of a reward. There are a number of issues that can limit the applicability of MARL techniques which we discuss next.

Problems of high-dimensionality

As the number of agents in these systems increase, there is a corresponding exponential increase in the possible communications and actions an agent may take with respect to other agents in the system. This increases the state-action space size, limiting the scale of systems that standard learning algorithms can be applied to. There are ways of mitigating this problem, and making large state-spaces tractable for computation.

State-space aggregation and abstraction

Through *aggregation* or *abstraction*, the number of states can be reduced through combining similar ones into a single state in the learning model[75], [76]. This simplifies the model, but sacrifices information about the merged states. Additionally, it can be difficult to qualify which states are similar enough to be abstracted, and the effect of doing so on the performance of agents can be unpredictable in more complex multi-agent systems.

State-space generation and adaptation

With *state-space generation* and *adaptation* algorithms, we can have the algorithm generate its own initial state-space representation[77], and then adapting this representation throughout its lifetime[78], [79]. This approach reduces the state-space down to those states relevant to an agent's learning function, while ignoring the others.

In Chapter 9 we introduce two algorithms that develop on this approach to tackle these problems caused by high-dimensionality. Our SAS-KR algorithm allows an agent to not only generate and adapt its known states through its lifetime, but also to forget state information that is judged to be less relevant. In addition, the N-Prune algorithm reduces an agent's state-space by restricting the number of other agents it can observe at one time, avoiding many of the challenges of large state-spaces.

3.6 Exploration strategies

Undirected and directed exploration

Finding the right balance of exploration, so that agents can discover the optimal actions in expansive state-spaces, and exploitation, so that they can successfully complete tasks, is difficult[80], [81]. Using *undirected methods*[82], where exploration is effectively at random, is not feasible in large state-spaces where the sparseness of action sampling slows learning. For this reason we focus on *directed methods* where knowledge can be used to make algorithms more selective in searching state-space.

Exploration in dynamic environments

The exploration/exploitation challenge for an agent increases in difficulty with the dynamism of other agents' policies and actions. In stationary environments, there is often an initial highly explorative stage, commonly using *ϵ-greedy* action selection[83], which then switches off in favour of a continual exploitation stage once the algorithm's performance is deemed to be acceptable. This may also take the form of a decay factor, where the degree of exploration decreases gradually over time such as in standard *Boltzmann exploration*[80]. In a non-stationary environment however, the tasks and their distribution may change. Agents may affect the environment and be affected by the behaviours of other agents. A fixed probability or time-based switch to exploitation risks a reduction in algorithm performance as the most optimal actions continue to change over time, but an agent's probabilities of choosing actions remain static.

Adaptive exploration

Adaptive exploration techniques[81] are designed for this non-stationary situation, varying exploration and exploitation throughout the system lifetime[84]. Other algorithms increase the exploration of infrequently sampled actions. Examples of this are count-based approaches[85], extending Boltzmann exploration with a state-action visitation factor[85]. Successor representations[86] have also been used as the state-action sampling metric to incentivise exploration[87].

While these methods can work in non-stationary environments where the degree of change is relatively constant, often the rate of change can accelerate or decelerate, or be relatively static in some areas of the system and highly dynamic in others. For example, in an ocean-based environment, currents might be volatile and rapidly changing in one part of the environment, but be stable with calm seas in another. To work in those environments, in Chapter 9 we introduce a variation of these approaches that utilises state-action space sampling history as well as past rewards history to guide exploration for our RT-ARP algorithm, discussed in Section 9.5.

No-regret exploration and intrinsic motivation

These exploration strategies use the principle of *optimism in the face of uncertainty*, the assumption that less well-known state-actions are worth exploring[88]. The use of *no-regret* to optimise reinforcement learning algorithms is well established[89] , with additional work applying this to exploration strategies[90], [91]. Agents can also be given different *intrinsic motivations*, underlying goals that generate rewards in addition to immediate task-completion benefits. Methods such as knowledge acquisition[92] or Bayesian curiosity[93] can then be used to drive exploration behaviours. Short-term and long-term intrinsic rewards can be combined to encourage local, and broader system exploration respectively[94].

In our work we adapt how optimistically an agent explores, not only based on uncertainty in the value of its actions, but on how optimally it believes it is exploiting the system given its past history. Whereas *advantage functions* (as used in actor-critic algorithms) compare the value of an action with those of actions that can be taken in the same state, we utilise the concept of neighbourhoods of agents[2], with some key assumptions on how these neighbourhoods change over time[3] to compare an agent's performance in its current state with its performance in similar, and highly dissimilar states in the past, using this comparison to influence its behaviour. The RT-ARP algorithm introduces a form of regret-minimisation exploration based on a function of the rewards over long and short-term timescales. We detail this work in Section 9.4.5 in which we describe how our *impact transformation function* is used by agents to predict the risk of taking more disruptive actions, and exploring more aggressively. This also provides a degree of risk-based intrinsic motivation, where agents are encouraged to explore more when their measures of short and long-term performance are unequal, and focus more on sampling-based Boltzmann exploration when they are comparable.

3.7 Challenges in large continuous systems

The stability-plasticity dilemma in continual learning

One of the challenges in non-stationary reinforcement learning is how much knowledge an agent should preserve about its previous experiences compared to adapting to more recent ones. This *stability-plasticity dilemma* affects how well agents complete new tasks they have previously seen[95], [96]. In the worst case it can result in *catastrophic inference*[97], where tasks an agent has completed in the past are treated as completely unknown when seen again in the future. The optimal balance of stability and plasticity is dependent on the proportion of tasks an agent sees in the future that will be similar to ones it has seen in the past. This is often achieved through *experience replay*, ensuring that past events are reapplied in the current learning context to not be completely overwritten by updates due to an agent's present actions[98]–[100] or localised learning updates to reduce overwriting past learned action probabilities[101], [102]. There are difficulties however in selecting which experiences are relevant in the present and should be reapplied. Successful or rewarding past actions may not be useful in an agent's current context given the non-stationary nature of the environment.

The RT-ARP algorithm introduced in Chapter 9 helps to address the stability-plasticity challenge by measuring an agent's performance over a range of short to longer-term timescales, then adapting the speed of the agent's learning based on the comparison

[2] See Section 8.2.

[3] See Section 9.4.5.

of task rewards over these periods, discussed in Section 9.4.5. The effect of this is that plasticity is increased. Behaviours are more strongly overwritten when the current policy of an agent is performing well in the short-term, but poorly over the longer term. As shorter and longer-term rewards become comparable, plasticity is decreased and learned values become more stable.

Coordination in agent-based systems In general, coordination in multi-agent systems increases the optimality of solutions found, but at the cost of increased overhead, which limits scalability. Agents in MAS can range from being fully cooperative to fully competitive. In cooperative systems the agents all share a common reward function and try to maximise that shared value function. Dedicated algorithms often rely on static, deterministic, or on exact knowledge of the states and actions of other agents. Coordination and maximisation of joint-action states results in high dimensionality due to the inclusion of the actions of other agents in calculations. To avoid this overhead, we can utilise the sparseness of the interactions in large multi-agent systems to reduce the coupling between agents by having them work independently and only collecting information about other agents when required. For example, by learning the states where some degree of coordination is needed[103]–[105].

Similarly, when approaching tasks that can be decomposed and allocated amongst a group of agents in a multi-agent system, we can use decomposed reward signals to induce some degree of coordination amongst localised agents that share those subtasks[106]. In a non-stationary environment, the value of those tasks to the allocating agent, and the capability of those agents completing subtasks, can change significantly. This means fixed rewards will lead to non-optimal coordinated behaviour. The combined output of our work in Chapter 9 is the ATA-RIA algorithm, designed to mitigate this problem. It enables an agent allocating tasks to continually vary individual decomposed reward values based on the outcomes of its allocation of subtasks to other agents.

3.8 Summary

This chapter went into detail on the task allocation problem in multi-agent systems, and highlighted research showing the key challenges we look to tackle in our work;

- in large or complex systems the correct policies for agents' behaviour are not known at system initialisation, and may be constantly changing due to system dynamics;
- since systems may be dynamic, the optimal solution may be constantly changing;
- for a system to be scalable, system-wide knowledge is not feasible to maintain or to compute with;
- agents have physical constraints on compute and memory in real situations that limit their maximum resource usage.

We briefly introduced some of the algorithms we present in Chapter 9 which are designed to tackle these issues. In the next chapter we look at the resource allocation and its part in optimising task executions.

Chapter 4

Resource allocation

Tasks may require resources to be used in their execution which must be provided by the agents who have been allocated those tasks. This chapter looks at how agents can optimise the allocation of their resources to achieve the best performance in completing these tasks (Contribution 2).

4.1 Introduction

In completing tasks or actions, agents need to utilise resources. These may be resources shared with other agents in the environment or resources that are dedicated to each specific agent. The resources required may be finite, being used up by the agent completing a task, or they may be unrestricted, but are only able to be utilised at a finite rate. E.g. a robot that makes welded repairs to pipes has a limited amount of solder to use before that resource is used up[107], whereas an agent monitoring ocean salinity may use up battery power in taking a measurement, but energy is continuously replenished through a solar panel[108].

Chapter structure

In the next section, Section 4.2, we briefly define the resource allocation problem. We look at some examples in different systems, and the key elements of the problem in Section 4.3. Section 4.4 then goes in detail to describe some algorithmic approaches to tackle the problem, and their limitations.

Example 4.1.1 (*Resource allocation in contamination monitoring*)**.** A set of sensors, each controlled by an agent, are deployed across an area contaminated with radioactive waste. The agents are repeatedly given tasks to measure the temperature, humidity, and radiation levels in their location. The more battery energy an agent dedicates to taking a radiation measurement, the longer the sampling time and the greater accuracy of the measurement. However, in doing so it has less battery power to dedicate to temperature and humidity measurements, providing less accurate readings.

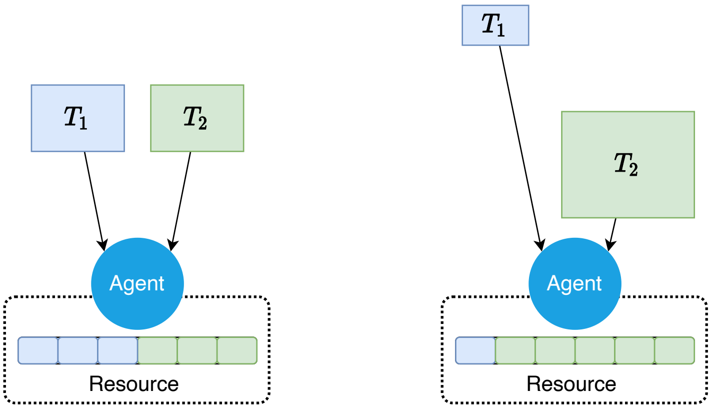


Figure 4.1: The allocation of resources to prioritise tasks. *An agent has been allocated tasks T_1 and T_2. Both require the use of the same resource, available in fixed quantities to the agent, however, T_2's completion is more valuable to the system's goal than T_1. In the first figure, the resource is shared equally amongst both tasks, which are completed at the same priority by the agent. In the second figure, the agent allocates more of its resource to T_2, which it now achieves better performance on. However, this now means that T_1 has a reduced amount of resource allocated to it and as a result the agent's performance on its completion is reduced. As T_2 is the more valuable of the two tasks, the overall utility of the system is increased.*

4.2 Defining the resource allocation problem

When allocated a set of tasks to complete, an agent may dedicate its available resources to these tasks in different quantities. In doing so, the *quality* of a task, the performance achieved by the agent in executing that task, can be increased. However, increasing the resources dedicated to the completion of one task can reduce the quality of others. As a result of these resource limitations, agents have a choice to make on how they allocate their resources amongst their different tasks, to optimise the outcome (See Figure 4.1).

The resource allocation problem

Definition 4.2.1 (*The resource allocation problem*)**.** The *resource allocation problem* can be defined as how to allocate an agent's available resources amongst a set of tasks that will optimise the utility in a MAS given that; the resources may be required for multiple different tasks, and the available amount of each resource may be fixed, or variable, over the lifetime of the system.

4.3 Objectives in multi-agent resource allocation

Examples demonstrating the *resource allocation* problem exist in many applications where the actions of agents place demands on shared resources, or in *task-prioritisation* problems, where multiple agents allocate tasks to an agent and compete for it to prioritise their task. Systems such as these are often categorised as examples of *multi-agent resource allocation (MARA)*[52] or *dynamic task-scheduling* problems[109].

Resources in V2X systems

We discussed the types of tasks being allocated inside a V2X system, such as requests for other agents' locations, or congestion information, in Chapter 3, Section 3.3. Each of these tasks requires the vehicle processing the request to dedicate resources to acquiring and aggregating information. Each of these tasks also has a varying degree of value to the vehicle receiving the data, depending on its situation. A vehicle travelling at speed may prefer nearby vehicles to provide position and velocity data. When vehicles are further away then less immediate factors such as traffic density in the area may become more valuable[110], [111]. A vehicle may receive such tasks from many vehicles and must decide how to prioritise and allocate the required resources amongst these competing demands. The interactions can also be highly dynamic. Nearby vehicles may communicate updates at a high frequency, becoming less frequent as they move further away. Interruptions to connectivity can come from buildings, other vehicles, and interference[5], [112]. With each vehicle providing its own set of resources, there is also the possibility of forming an ad-hoc distributed compute platform amongst multiple vehicles, which requires efficient resource allocation to handle the many distributed tasks in the system[113], [114].

Key requirements

These examples highlight the key elements of the systems we look to provide a resource allocation solution for;

1. there is a distribution over time of incoming sets of tasks, which require decomposition and prioritisation by agents;
2. the value of each task is dependent on the unknown state of the agent that requests its completion;
3. there are multiple competing tasks demanding resources for their execution;
4. the types of tasks agents receive may vary over time, as well as the value of those tasks.

4.4 Algorithms for resource allocation

There are established methods that can be applied to the resource allocation problem such as auction protocols and automated negotiation schemes[51], [115], [116], e.g. contract-net[117] and its more recent extensions[118]. In these, an agent announces the availability of some resource it owns, agents interested in gaining access to the resource make bids, then the resource owner makes the final allocation.

Scalability of auctions and negotiation

Although negotiation algorithms have been developed that are broadly used in many areas, the limitations become apparent when applied to large-scale and complex systems in particular[119]. The first problem occurs due to the *lack of scalability* with increased number of participating agents. When there are many agents bidding or allocating resources, the negotiation process can carry a significant overhead. This can delay, or otherwise reduce the optimality of, the overall allocation of resources. Extensions such as concurrent contract-net[53] mitigate some of those effects by extending the negotiation protocol but inherently have the same limitations.

Resource allocation complexity

The second issue comes from the effects of *resource allocation complexity*. The allocation of resources to one agent can positively or negatively impact further agents whose own demands rely on that agent's ability to acquire those resources. In addition, where there are many agents requesting a resource-type, and many that can allocate that resource, finding the optimal distribution of those agents' demands across those resource-allocating agents is a challenging problem. In some cases *combinatorial auctions*[120], [121] can help develop more complex allocation strategies, but with an associated negative impact on the scalability of solutions.

Joint and independent action learning

Longer-term effects, and more complex relationships, can be modelled through *multi-agent reinforcement learning (MARL)*[71] to learn, and adapt, resource allocation policies. Broadly speaking there are two main strategies that can be adopted; *joint-action learning*[122] learns a model for the system as a whole, using the combination of knowledge from all agents in the system; and *independent-action learning*, where each agent learns independently of other agents in the system using only a localised view of the system[122], without coordination. Examples of joint-action algorithms can be found in Q-learning[123], [124] and deep learning[125]–[127], and some independent-action based solutions based on distributed Q-learning algorithms [128], [129]. We will cover some state-of-the-art algorithms in Chapter 5 that often use combinations of both strategies to achieve the best performance.

Challenges in joint and independent-action learning

Whereas we can best solve the problem of global resource allocation optimisation through joint-action learning, this becomes computationally intractable as the number of agents and states increases, and so suffers from the scalability problem. Alternatively, independent-action learning avoids the costs of intercommunication with other agents so is more scalable, but without a system-wide view does not optimise as well. This is due to the lack of observability of other agents' strategies, and risks becoming stuck in locally-optimal solutions[109], [116], [130]. However, recent work utilising on-policy reinforcement learning algorithms such as *proximal policy optimisation (PPO)*[131] based approaches has shown that reinforcement learning can be practically effective in some independent-action learning systems [132], [133].

Centralised training with decentralised execution (CTDE)

Centralised training with decentralised execution (CTDE) algorithms[134], [135] look to combine the best of joint and independent-action approaches. Agents develop local policies, however during a training phase there is centralisation using shared, system-wide information to learn the best policies. For example, multi-agent deep deterministic policy gradient (MADDPG)[136] has decentralised agents, learning a local policy from their own observations, with a centralised critic that can use information from all agents. After an initial training period, the agents' localised actors are used in isolation[1]

Providing allocation feedback

As we develop our algorithms in Chapter 10, the approach we will follow is to extend an agent's localised view by enabling a *parent agent*, the resource-requesting agent, to feedback to a *child agent*, the resource-allocating agent, the value of the child agent's allocation strategy to the parent's broader goals. This passes information to the child agent on how its allocation may have affected other child agents' task completion qualities. Through this extension of a localised learning approach we are able to model the complex outcomes involved in multiple agents' resource allocations. However, as each agent's view of the system is still constrained, it scales to large systems.

[1] See Chapter 5, Section 5.2, 'Actor-critic methods' for more detail.

4.5 Summary

In this chapter we looked at the problem of optimising resource allocation, the drawbacks that can come from using auction-based algorithms, and the differences between agents being able to solve the problem jointly, or independently.

Chapter 5

Reinforcement learning in agent systems

In this chapter we introduce reinforcement learning methods, and how they can be used to help achieve our research goals. We examine the ways these techniques are applied to agent-based systems, and the difficulties found in doing so.

5.1 Introduction

Reinforcement learning covers a class of problems where an agent in a state takes an action and adapts its future behaviour in response to a reward signal from the environment. Using a number of different algorithms and conceptual approaches from this area there are many complex learning problems that can be successfully tackled[137]–[139].

Chapter structure

In Section 5.2 we look at the basic concepts for learning an environment, and how to explore that environment in Section 5.3. We briefly introduce some dynamic programming techniques that will be used later by our algorithms in Section 5.4 and key considerations around balancing learning policies in Section 5.5. We focus in more depth on multi-agent reinforcement learning (MARL) in Section 5.6, with consideration of the challenges to successfully applying these algorithms in Section 5.7.

5.2 Modelling the system

Learning from experience

An agent's *policy* is a deterministic or stochastic mapping that relates states in an environment to a set of actions that the agent should carry out in those states. This policy defines the agent's behaviour in a system. Policies can be learned and adapted over time as the agent takes actions, and experiences the outcomes of those actions. *Rewards* enable the agent to receive feedback from the environment on the success of its policies, driving its adaptation. It is through this process that the agent learns increasingly beneficial policies to pursue.

Value functions The accumulation of rewards over time through taking an action is its value, often estimated through a *value function*. This encourages agents to make better long-term decisions in contrast to purely short-term ones driven by immediate rewards. The estimation of value functions and the constant refining of them given actual outcomes is central to the success of reinforcement learning techniques.

Modelling the environment In learning an optimal strategy there are two main approaches agents may take. In *model-free* learning an agent has no internal map of the environment, and takes a trial-and-error approach to learning better policies, simply reacting to the feedback it receives as a result of its actions. However, an agent could instead attempt to predict the outcome of its actions, which requires it to model the environment, giving *model-based* learning. Using this approach allows an agent to reason about planning, before selecting actions. In our work we focus on model-free algorithms as the systems we will be targetting are unknown and the optimum behaviour uncertain.

Model-free algorithms fall into two main categories of reinforcement learning algorithm for agents; *value-based reinforcement learning*, where agents learn the value function directly, using the expected cumulative reward of taking an action in a state to infer an action selection policy; and *policy-based reinforcement learning*, where the agent directly learns a mapping from states to actions without explicitly using a value function, instead optimising the policy by maximising rewards through the use of techniques such as gradient ascent.

Comparison of value-based and policy-based approaches These two reinforcement learning approaches come with their own strengths and weaknesses[140]. Value-based methods store no model and instead combine past information into simple scalar values reflecting how good an action in a certain state is from past trials, making them often simpler to use and more explainable. They can store past information learned about the environment explicitly and search for the best current policy, but at the cost of increasing computational complexity. These methods often require a lot of trial and error to learn and adapt to changes in the environment[141]–[143].

Policy-based approaches come with their own challenges in MARL. They inherently suffer in non-stationary environments, where learning instability is often increased and enhancements such as *experience replay* become less applicable[136], as well as often showing high variance[144] (which can be tackled through actor-critic methods). However, they can be effective in high-dimensional or continuous action spaces where value-based methods scale poorly.

Actor-critic methods

Reduction of the variance of solely policy-based algorithms can be achieved through *actor-critic based reinforcement learning* methods[145]–[148]. This strategy combines a policy-based actor, which selects the actions that dictate an agent's behaviour, and a value-based critic, that judges how good that action was (used in the update to the actor's policy). Such methods indeed reduce variance, which can be improved further through the use of an *advantage function* as critic instead of the value function, so that the action chosen at a state is compared to the average value of the state[149]–[151]. However, drawbacks such as the sample-efficiency of policy-based methods and the bias introduced by actor-critic methods, are still considerations in their application.

The use of value-based, policy-based and actor-critic based reinforcement learning approaches can be found in recent state-of-the-art algorithms[152], [153]. For examples, see the value-based deep-Q network (DQN)[154], and policy-based trusted region policy optimisation (TRPO)[155], and actor-critic based deep deterministic policy gradient (DDPG)[156] algorithms, as well as extensions of these, and others strategies, in Tables 5.1, 5.2, and 5.3.

In our work we focus on improving the performance of value-based approaches in non-stationary environments so that they can be utilised in a greater range of systems where they previously may have been inapplicable or performed poorly.

5.3 Exploration and exploitation

An agent should not always take what it believes to be the best action in a given state, as it may not have learned enough about the possible actions in that state to have found the optimal one. Or indeed, the best action to take may change as the environment evolves over time. The problem is that the agent may have a very accurate estimate of the value of taking an action due to sampling that particular one many times, but, having not chosen the alternative actions in that state very often, very poor estimates of those other actions that may actually lead to better outcomes.

In addition, we have to view the result of actions over the long term. For example, taking an action with a low reward could lead to states where there are actions with high overall value yet to be explored by the agent. It is this balance between *exploitation*, where an agent takes the current optimal action and gets the expected highest reward, and *exploration*, where it has more uncertainty about the reward, that ensures that the agent investigates the solution space for the best overall policies.

In order that the agent does not get stuck persistently taking suboptimal actions, and explores other actions about which it has more uncertainty, we allow for some probability that the agent will refuse to take what it believes to be the optimal action, and take another instead. An agent can use simple methods such as probabilistically taking random, possibly suboptimal actions as used in ϵ-greedy exploration[157], and other, more complex adaptations of this approach[158]. Annealing methods use a time-dependent probability distribution to select actions based on their predicted outcomes, such as in the commonly used *Boltzmann exploration*[159] method. *Intrinsic curiosity*[160], [161], or *intrinsic rewards*[162], can push agents to explore based on some quality internal to the agent, rather than an external reward signal. Other methods prioritise exploring the unknown, sparsely sampled areas of state-action space more effectively and efficiently[163].

Our work on task allocation, described in Chapter 9, utilises both ϵ-greedy and intrinsic reward approaches for adaptive exploration based on an agent's belief about its current performance.

5.4 Dynamic programming

For an agent to discover its optimal policy it needs to calculate the value of cumulative rewards for the actions it can take. This could mean taking into account actions and rewards from $t = 0$ to the current time, which quickly becomes intractable. If we assume the *Markov property*, that the probability distribution of future states depends only on the current state and not on any previous ones, we can apply methods from dynamic programming such as policy, or value iteration to find solutions. As these methods depend on knowledge of action probabilities and state transitions they are not directly applicable in systems where agents need to learn these from interactions with the environment. However, they have strongly influenced the development of reinforcement algorithms that are used in their place.

Markov Decision Processes (MDP) A Markov decision process models the situation where an agent in state s takes action a and moves into the state s', therefore receiving a reward r. Using this definition the probability of moving from the current state-action pair $\langle s, a\rangle$ to a new one $\langle s', a'\rangle$ is:

$$P(s', r|s, a) = Pr\{s_{t+1} = s', r_{t+1} = r|s_t = s, a_t = a\} \tag{5.1}$$

The goal for an agent is to find an action policy $\pi(s)$, for the state s, that maximises the accumulated reward. If we apply a discounting factor γ, then the overall sum of rewards is:

$$\sum_{t=0}^{\infty} \gamma^t r_{a_t}(s_t, s_{t+1}) \tag{5.2}$$

Where a reward r_{a_t} is received by an agent taking an action a_t at time step t.

Where the agent does not have certainty on the current state of the system we have a *partially observable Markov decision process (POMDP).* In this case, when an agent takes an action in state s it does not deterministically end up in state s'. Instead there is a probability distribution over possible subsequent states from s and the agent instead maintains beliefs on the current state given any observations it has of the system.

Temporal Difference Methods Combining Monte Carlo methods and dynamic programming, *temporal difference learning*[164] updates an agent's current prediction of the estimated value of taking an action based on its estimate of the value in the next time-step. This means that the estimates of value returns can be updated in a *bootstrap* fashion using other estimates without waiting for actual values to be returned[165]. The estimated value can be updated using the value estimate of the next time step $\hat{V}_t(x_{t+1})$ with discount γ, such as in the *temporal value difference update*, $r_t + \gamma\hat{V}_t(x_{t+1})$.

Using *eligibility traces*[166] we can update values based on estimates across multiple states and time-steps. This gives a family of temporal difference functions where $TD(0)$ represents updating only the previous prediction all the way to $TD(\lambda)$ where all previous predictions are updated[167]. For example, for the accumulative eligibility,

eligibility tracing

$$e_t(x) = \begin{cases} 1 + \gamma\lambda e_{t-1}(x), & if\ x = x_t \\ \gamma\lambda e_{t-1}(x) & otherwise \end{cases} \tag{5.3}$$

Eligibility tracing will form part of our solution to resource allocation in Chapter 10, Section 10.3.2.

Q-learning[168] is an *off-policy*, model-free approach that learns functions of Q values, action-values that describe the utility of the resulting outcome given optimal choices thereafter. $Q(s, a)$ is the Q-value mapping of a state-action pair to a value, an agent's estimate of the future reward of taking action a in state s. The update uses the current Q-value and the reward for taking the action a in state s and arriving in the next state s'. The learning rate $\alpha \in (0, 1]$ and the discount factor $\gamma \in (0, 1]$ control the rate at which the learned values are updated:

Q-Learning

$$Q(s, a) \longleftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma maxQ(s', a')] \tag{5.4}$$

Where r is immediate reward that the agent receives after taking action a in state s and $maxQ(s', a')$ is the current estimate of the optimal Q-value that can be obtained in the next state s'.

In Chapter 9, Section 9.4 we detail how we use Q-learning as part of our solution to the task allocation problem, and how we adapt to handle non-stationary environments through a form of intrinsic rewards dependent on an agents belief of its current performance.

5.5 Balancing Exploration and Exploitation

There are a number of common algorithms used to balance the needs of an agent to exploit its perceived optimal action and exploring new pathways through the policy space.

The greedy policy is simply to always take the maximal rewarding action a in any given state. This means that there will be no exploration of uncertain or non-optimal paths. So, given a state s with possible set of actions A_s available to the agent in that state:

The Greedy policy function

$$a = \underset{b \in A_s}{argmax}\ Q(s, b) \tag{5.5}$$

If instead of choosing an action with a greedy policy we generate a small probability $0 \leq \epsilon \leq 1$ that we choose another action randomly from the set of possible actions in state s we can introduce an element of exploration into our strategy. So, if for each time step we generate a random number $0 \leq \xi \leq 1$ then:

The ϵ-greedy function

$$\pi(s) = \begin{cases} random\ a \in A_s, & if\ \xi < \epsilon \\ \underset{b \in A_s}{argmax}\ Q(s, b), & otherwise \end{cases} \tag{5.6}$$

The Boltzmann function One issue with ϵ-greedy methods is that all non-optimal actions are treated equally. The agent will be as likely to explore what it believes to be the worst possible action as it would a close-to-optimal one. *Softmax functions* help with this by making the choice of exploration action dependent on the current estimates of their value, meaning that high-valued actions are more likely to be chosen for exploration than low-valued ones. The *Boltzmann function* is the most common algorithm for balancing exploration/exploitation this way. Similar in concept to that of simulated annealing, the defining characteristic is that of the temperature of the computation τ. This variable dictates how equal the choice of actions will be. With a high value, all actions will be treated as equally probable choices. The lower the value, the more the preference for higher valued actions.

$$P(a|s) = \frac{\exp(\frac{1}{\tau}Q(s,a))}{\sum\limits_{b \in A_s} \exp(\frac{1}{\tau}Q(s,b))} \tag{5.7}$$

5.6 Multi-agent reinforcement learning (MARL)

MARL[71], [73], [74], [169] applies reinforcement learning techniques to systems where many agents share a common environment, where an agent taking an action causes a transition of the system state to a new one. There are challenges in getting the right exploitation/exploration balance, as well as coping with the rapid increases in state-action space size that come with realistic systems.

Curse of dimensionality The dimensionality explosion that occurs in state-action space for single agents is even more prevalent in the multi-agent case. An agent may not only have to learn about its own effects on the environment but also about the nature of other agents in the system. The exploration/exploitation issue increases in difficulty as now there is not only the question of the environment being stationary but also having to learn and adapt to the dynamism of other agents' policies and resulting actions changing the system state. With rewards possibly being an outcome of multiple agents' policies there is also uncertainty in how to share those rewards fairly when the contributions of individual agents are not always easily differentiable, or clearly correlated with a single action.

Cooperation and competition MARL systems fall somewhere across a spectrum defined by full agent cooperation on one end and agent competition on the other. In fully cooperative systems the agents all share a common reward function and seek to maximise a shared value function. Algorithms dedicated to these cases tend to rely on static, deterministic, or exact knowledge of other states or agent actions. Due to the need for coordination and maximisation of joint-action states many algorithms suffer from computational complexity stemming from high dimensionality. Tables 5.1, 5.2, and 5.3 summarise some of the algorithms designed to work in a range of multi-agent learning systems. For a comprehensive taxonomy see Zhang et al[170][1].

[1] See also Reinforcement learning, comparison of reinforcement learning algorithms [171] from Wikipedia, and 'Spinning Up in Deep Reinforcement Learning'[172] from OpenAI

Method	Authors	Description
Team Q-learning	Littman (2001) [173]	Fully-cooperative with no coordination. Utilises joint action learning so number of states quickly becomes large as agent numbers increase.
Distributed Q-learning	Lauer et al (2000) [128]	Policy free system that makes an 'optimal assumption' that an agent's teammates complete what it considers optimal actions. Even in a deterministic environment a central plan is needed to provide some consistent coordination.
Optimal Adaptive Learning	Wang et al (2002) [174]	Biases so that optimal Nash equilibria are eventually selected. Provably converges to an optimal Nash equilibrium in any team Markov game. Highly complex and requires static games. Needs a model of agents, the game, and each stage.
Minimax-Q	Littman et al (2001) [173].	In a competitive system an agent using this algorithm will maximise its benefit under the assumption that the opponent will always act to minimize it.
Correlated equilibrium Q-learning	Greenwald et al [175]. (2003)	Uses the correlated equilibrium generalisation Nash equilibria. Has four main forms utilitarian (uCE-Q), egalitarian (eCE-Q), republican (rCE-Q) and libertarian (lCE-Q). Shown to empirically convergence to equilibrium policies in general-sum Markov games.
Deep Q-Network (DQN)	Mnih et al (2015) [154]	Q-learning where Q-tables are replaced by 2 neural networks.
Double Deep Q-Network (DDQN)	Hasselt et al (2015) [176]	Extension of DQN that tackles overestimation by using a second deep network to select actions.
Dueling DQN	Wang et al (2016) [177]	Extension of DQN that separates state-value and action-advantage functions.
QMIX	Rashid et al (2018) [125]	Monotonic value function factorisation approach that learns a centralised Q-function by using a mixing network that allows for coordination between agents while preserving individual control.

Table 5.1: Examples of value-based MARL algorithms (value-based). *A table covering some of the key value-based reinforcement learning algorithms developed in recent decades.*

Method	Authors	Description
WoLF-IGA	Bowling et al (2002) [178]	A policy search algorithm using Infinitesimal Gradient Ascent (IGA) combined with Win-Or-Lose-Fast(WOLF). The step size is small with a slow learning rate when payoffs are high, with a large step size when payoffs are low, increasing the learning pace.

continues on the next page...

...continued from previous page

EXORL	Suematsu et al (2002) [179]	Uses a policy that biased as to minimise another agents drive to change its current policy.
GIGA-WoLF	Bowling et al (2004) [180]	Keeps track of the agent's regret compared to playing a stationary pure strategy and guarantees that regret will be positive in the long run.
WoLF-PHC	Busoniu et al (2010) [73]	The combination of a Q-learning update rule and the gradient-based policy update from WOLF-IGA.
Trust Region Policy Optimization (TRPO)	Schulman et al (2017) [155]	Improves the sample efficiency and stability of reinforcement learning by constraining the policy updates to a trust region.
Proximal Policy Optimization (PPO)	Schulman et al (2017) [131]	Extension of TRPO that uses clipped surrogate objective to improve stability.
Independent proximal policy optimization (IPPO)	DeWitt et al (2020) [132]	Decentralised PPO algorithm where each agent estimates its local value function and learning is done independently.

Table 5.2: Comparison of MARL algorithms (policy-search). *A table covering some of the key policy-search based reinforcement learning algorithms developed in recent decades.*

Method	Authors	Description
WoLF-PHC	Busoniu et al (2010) [73]	The combination of a Q-learning update rule and the gradient-based policy update from WOLF-IGA.
Deep Deterministic Policy Gradient (DDPG)	Lillicrap et al (2015) [156]	A model-free off-policy actor-critic algorithm that uses deep neural networks to learn a continuous policy and Q-function in a continuous action space.
MADDPG	Lowe et al (2017) [136]	Extension of DDPG that learns decentralized policies for multiple agents in a centralized training setting.
Counterfactual multi-agent policy gradients (COMA)	Foerster et al (2018) [181]	Centralised critic with decentralised actors, incorporating counterfactual estimates of the impact of an agent's actions on the global reward.
Decentralized policy optimization (DPO)	Su et al (2022) [182]	A decentralized actor-critic algorithm with monotonic improvement and convergence guarantees.

Table 5.3: Comparison of MARL algorithms (actor-critic). *A table covering some of the key actor-critic based reinforcement learning algorithms developed in recent decades.*

Stability

In stationary environments we can assume that the same state-action pair will lead to the same probability distribution over outcomes. But in multi-agent systems this is no longer true. Each agent is continuously adapting its policies, so the environment is by its very nature dynamic. We therefore need to balance *stability*, where the system converges towards a solution, and *adaptation*, where an agent can react to the changes in the policies of other agents.

Convergence

We can approach this by using stationary strategies as targets, isolating any dependence on other agents' policy changes. In doing so, we require an agent to have the property of *rationality*, where it will converge to the best response under the assumption that other agents in the system are stationary. One additional requirement if the agents are non-cooperative is that this solution holds true no matter what strategies the other agents choose. This requirement is added to avoid an agent being deceived by other agents into choosing a poorly performing strategy.

Regret

Another approach is to identify the least *regret* in choosing a policy, in effect the difference between the performance of the dynamic strategy as compared to the best possible static strategy it could choose. An example of this approach is GIGA-WOLF[180]. This uses a Win or Learn Fast (WOLF) strategy[178], [183], which increases the learning step-size when the algorithm judges itself to be losing as compared to the equilibrium strategy. This is combined with the *generalized infinitesimal gradient ascent (GIGA)* algorithm[184] which uses the greedy projection of an unconstrained gradient ascent step onto the restricted space of constrained policies. GIGA allows for the calculation of the *regret* of the step.

$$\mathcal{R} \leq \sqrt{T} + |A| r_{max}^2 (\sqrt{T} - \frac{1}{2}) \tag{5.8}$$

GIGA-WoLF allows us to compare two strategies and increase the learning speed to drive an adjustment to the first to bring it closer to the second only if the second strategy is the better one.

Coordination

There are areas of research that look to mitigate the increased complexity of computation resulting from agents coordinating their actions. By utilising the sparseness of interactions in large multi-agent systems, the coupling between agents can be reduced by having them work independently and only collecting information about other agents when required. For instance, by learning the states where some degree of coordination is required[104], [185], or through the use of *curriculum learning* methods, where the agents are trained on simpler tasks before being introduced to more complex tasks and systems[186], [187]. E.g., learning in a single-agent environment before translating those policies into the multi-agent environment for further reward-driven adaptation.

5.7 Challenges in real-world environments

Simulations have the benefit of being able to run repeatedly, learning in controllable, virtual environments. In the real-world however, algorithms must adapt to noisy information, and less reliable interactions with the environment. Robotics provides examples of common challenges in real-world reinforcement learning problems. These systems are often noisy and partially observable with a large, stochastic state-action space. Agents are not certain of the state-space they are in and instead must work with belief-states based on their understanding[188]. The speed of execution of learning algorithms becomes of practical concern with real-world learning often being time-consuming, expensive, with real-time needs or physical risks in decision choices[15].

Differentiation of states One of the main issues arising in real-world systems is that of the exponential increase in state-action space dimensionality. States that appear equivalent in an idealised system, are in reality a multitude of distinct states due to the existence of abstracted away dimensions in theoretical or simulated systems. In a networking simulation for example, we might assume that temperature was a constant across our virtual network. In practice, the existence of slight spatial differences in temperature lead to a differentiation of states and a large expansion in state-space. There are strategies that enable us to reduce real-world dimensions down to lower-dimensional state-space through dimensionality reduction techniques such as *principal component analysis (PCA)*[189] and neural network pre-processing of data[190]. However, the loss of information and possible convergence on poor policies is a risk.

Sparse sampling Real-time interactions provide less opportunity for agents to learn from taking actions, so understanding the impact sparse sampling can have on the quality of estimated learning policies becomes important[191], [192]. Sometimes there are expensive and fragile hardware components at stake, such as in robotics, therefore interactions must be actively reduced to avoid wear. There may also be dangerous states and actions that need to be precluded from learning to prevent damage or dangerous outcomes[193].

Simulation to reality gap We can utilise the accelerated learning times of simulations to build up agent policies that can be transferred across into real environments. The success of this approach is greatly dependent on the stability of these learned policies under perturbation, where instability, inaccuracies in sensor data, lack of granularity in the model, or abstracted-away environmental noise, can make transferring the policies very brittle[194].

Reward sparsity Providing a good reward function for successful learning has proven to be a complex task[195]–[197]. If a simple reward is provided, such as for success or failure in completing a task, but the task itself is complex, then the agent receives rewards infrequently and policy learning can be extremely slow. This *reward sparsity* can be mitigated through the use of methods such as knowledge transfer[198], curiosity-driven rewards[199], curriculum learning[200], and experience replay[201], to enhance the frequency of rewards.

Experience replay Experience replay has recently seen much development and application to deep learning systems[202], [203]. Past strategies have been extended to be more stable in large, non-stationary systems[204] and to provide reward shaping through the generation of sub-goals from past experiences captured in a replay buffer[205].

Reward shaping

In order to provide more regular feedback we can use *reward shaping*, where intermediate rewards are provided that indicate desirable steps towards achieving a goal. For example, a mobile robot could be given a series of rewards for reaching waypoint markers as it moves closer to its final destination. Beyond manual reward shaping *inverse reinforcement learning* is also possible, providing shaped rewards by having agents learn by demonstration[206].

We can also attempt to have the agent learn to decompose its overall goal into subgoals using various strategies. One approach uses differentiation of the problem into two separate goals, a task-specific one, and a task-agnostic one. The former being used to learn specific tasks with the latter being used to identify useful patterns in solving tasks that can be used as dynamically generated sub-goal rewards[207] . This does however introduce a significant design problem in setting out what that agent-space might be and how it might be affected by sub-goal patterns that restrict the exploration of problem space to non-useful areas[208].

Hierarchical reinforcement learning

The complete task can be broken up into smaller subtasks that we know will contribute to a successful outcome through *hierarchical reinforcement learning*[209]. Similar to reward shaping but more direct this again restricts the exploration of state-space by constricting the paths through it to a successful outcome around predefined localities. An example of this might be decomposing a walking task into multiple subtasks that move each leg in coordination. Other approaches utilise a manager-worker model[210], where a 'manager' agent decomposes its own reward function into subgoals and rewards for 'worker' agents. Aspects of our work has similarities to this approach, where parent agents decompose their task into subtasks for child agents, and allocate the resulting reward to those child agents based on the overall outcome of the parent agent's task[2].

Reducing state-action space complexity

To reduce complexity and make learning tractable we generally look to reduce the size and dimensionality of state-action space. By doing so, calculations become faster and searching for the best actions is done in a less expansive policy space. One simple approach is to make the continuous environment less granular by *discretisation*, either by segmenting some of the dimensions of state-space[211], or by allowing an adaptive granularity of those dimensions dependent on the learned importance of localities within them[212].

Knowledge reuse

Learning can be accelerated by providing some form of *knowledge reuse*[213]–[215], reducing the search space for optimal policies by guiding algorithms into known rewarding areas. Demonstration techniques such as *apprenticeship learning*[216] use expert knowledge to lead the agent through the task directly. *Imitation learning* can train agents by having them replicate human behaviour on a task[217], or that of other agents using *inter-agent learning*[218]. Using these strategies, policies can start in areas of policy space that have already proven to be successful, avoiding the agents starting from scratch. The current research also illustrates the difficulty in carrying out learning transfer in MARL systems. There needs to be an understanding on what depth and detail of knowledge to transfer and how to map it to new situations. In systems where tasks or roles change with significant frequency this is essential to prevent constantly learning from zero knowledge, which degrades system performance and convergence[219].

[2] See Chapter 9, Section 8.3.

Our work in Chapter 9, in particular on the state-action space knowledge-retention (SAS-KR) algorithm[3], demonstrates a form of inter-agent learning. Agents retain prioritised knowledge of other agents that helped them to perform their tasks better, and can send this information to other agents on request. This enables the agents that receive this information to reuse that knowledge to improve their own task performance, although the agents learned about from shared knowledge may not be as useful to the requesting agent as it was to the agent that supplied the information.

5.8 Summary

In this chapter we covered some of the main techniques using reinforcement learning in multi-agent systems. We also highlighted where compromises in practicality and response-time need to be balanced when applying algorithms to both simulation and real-life systems. A major issue that needs to be considered is how computational complexity is affected by the 'curse of dimensionality' that is a direct consequence of having multiple independent agents with their own policy spaces working simultaneously on related goals.

The ability to reduce down the space of other agents and interactions to be considered for an agent is clearly of benefit here. By only considering a subset of other agents in the system, or only the information necessary to complete the required tasks successfully, we can greatly reduce the dimensionality of the problem, make the systems more scalable, and work towards real-time responsiveness.

[3] See Chapter 9, Section 9.5.

Chapter 6

Self-organisation in agent-based systems

In this chapter we look at self-organisation and emergent behaviours, how structure can form in multi-agent systems without a fixed, pre-determined architecture. This is relevant to our research work in how agents form different roles while participating in the completion of tasks (Contribution 3).

6.1 Introduction

To orchestrate the completion of tasks, a system often requires some degree of organisational structure and communication. For example, for an agent to discover the capabilities of other agents to carry out a given task, or to discover more agents in the system to collaborate with. Ways of achieving this broadly fall somewhere on a spectrum. At one extreme, organisational structure and roles are defined at design time, with agents in the system able to reorganise within the constraints of the structure set out, and the roles available, to adapt the system during runtime. At the other extreme, there is little or no prior specification of organisational structure at design time, instead the structure, and the roles agents perform, are part of the emergent behaviour of the system.

Chapter structure

We look at how to define and categorise self-organising systems in Section 6.2 and their properties in Section 6.3. How we can measure self-organisation and its effectiveness is the topic of Section 6.4, followed by a brief introduction to holarchy as a method of understanding self-organisational structure in Section 6.5.

6.2 Defining of a self-organising system

We can describe the method by which MAS can autonomously alter their structure as *'...the process by which a system changes its organisation at runtime without an external command'*[25]. However, we need to differentiate between two closely related approaches to organisational change.

Organisational-centred point of view The first comes from a top-down view of the system, the *organisational-centred point of view (OCPV)*. From this perspective the system is seen as a designed organisation, with the agents constrained or defined by the overarching organisational specifications and schema. This means that agents can be aware of the organisational structure at run-time as it is formally set out. With this knowledge they can reason about the current state of the system and enact changes to adapt its structure, such as to adapt roles, communication patterns, destroy or create new agents, and join or leave social groupings. They cannot however create new structures outside of those prescribed by the designer. This type of system is a *reorganisational system*[220], [221].

Agent-centred point of view From the opposite side of the spectrum comes the bottom-up behavioural viewpoint, the *agent-centred point of view (ACPV)*. In this case, the system's structure is the result of agents' local actions driving the emergence of higher level organisation and behaviour. Since there is no prescribed model of the organisation as a whole, agents are not aware of global structure, and cannot make the same level of reasoning as made in OCPV systems. They can however build internal models using their partial-view of the system, and use these models to reason about local behavioural patterns and social structures. When environmental pressure is felt by the system, agents enact adaptations in their local vicinity, and so organisational changes propagate through the system resulting in global structural change. This localised, bottom-up behaviour distinguishes these systems as *self-organisational systems*[37].

Strong and weak self-organisation Within self-organisational systems, we can also categorise by the strength of internal decentralisation of organisational change[222]. Systems with *weak self-organisation* have some degree of internal, centralised control, as exemplified by a queen ants control over an ants nest. *Strong self-organisation* however involves neither external nor internal centralised control of any kind. Given our target systems, we are focused on the strong self-organisation case for developing our algorithms in Part II.

6.3 Properties of self-organising systems

Self-organising systems share some common characteristics. There must be no external control over the system, and the internal controls and information flow must be decentralised. In addition, the system must be adaptable, in that this property will be necessary for actual self-organisation during runtime to take place. We summarise the core characteristics below, adapted from Serugendo et al[25], as well as how they apply to our target systems.

- *endogenous global order*, the internal behaviour of the system drives it towards a stable, global state. With possible variations in incoming task distribution and changes to the environment in our dynamic MAS systems, the optimal global structure itself may be dynamic. However, we make the assumption that in our systems, the rate of change in optimal structural is small compared to the ability of agents to adapt.
- *emergence*, local interactions produce the global behaviour, with no central control driving this behaviour. The local interactions between individual agents that generate these global effects cannot be directly related to observed local actions.
- *simple local rules*, there are simple rules that control the behaviour of individual

agents within the system. These rules do not contain enough information to describe the overall global behaviour of the system. Instead, they only have enough information contained in them to describe the individual behaviours which will result in the generation of the global behaviour of the system.

- *dissipation*, without external changes, the system is expected to reach some stable global state rather than continuously changing. However, the systems we study do expect some external variation in the distribution of incoming tasks during the systems lifespan. There will be changes in agent connectivity due to the environment, and changes in population make-up due to agents leaving or joining the system.
- *instability*, the system's behaviour is non-linear and sensitive to initial conditions and parameters, meaning that we can not simply examine the individual components at system initiation and so understand the system's future states.
- *multiple equilibria*, as agents can adopt a range of roles within the system, there may be many different configurations of their actions, task allocations, and roles that will meet the system's goals. This means there may be multiple equilibria that the system could be in while still meeting its goals.
- *criticality*, we see phase changes and threshold values within the system.
- *redundancy*, the system is resilient to damage in that agents' capabilities overlap to the extent that the loss of an agent would still allow the completion of the allocated tasks.
- *self-maintenance*, the system can self-heal. With damage to the system or communications the system can introduce new agents, or route around communication disruption.
- *adaptation*, the system can adapt as the environment changes.
- *complexity*, the system cannot be defined globally in terms of simply a combination of local agent behaviours.
- *hierarchies*, there are multiple levels of self-organisation. Agents may be grouped into nested hierarchies or roles in task completion.

6.4 Measuring self-organisation

There are a number of useful metrics in analysing a self-organising system that arise from the three facets of the system, the organisational structure, the process that produces and maintains it, and the function the system is aiming to fulfil[25]. Firstly, *convergence time*, the time taken until convergence to an organisational structure where the system is able to fulfil its overall goal. Next we have *perturbation recovery time*, the time taken to reorganise after perturbations, and the related quality of *perturbation resilience*, the stability and adaptability in the face of such perturbations[223]. Finally, the *degree of decentralisation of control*, where on the spectrum from fully externally designed to purely autonomous self-organisation the system lies. We will use the metrics of convergence, perturbation recovery, and perturbation resilience to measure our algorithms effectiveness in Chapters 9, 10 and 12, later in the thesis. As our targeted systems are fully self-organised and decentralised, the degree of decentralised control metric is not relevant.

There are known practical examples of self-organisation in areas such as adaptive meshing in cellular networks, traffic simulation, and flood forecasting[224]. There also exist mobile robotic applications utilising swarm behaviour[225], [226].

6.5 Holarchy

A *holon* is defined as something that is simultaneously a whole and a part[227]. When layers of holons occur in a system we have can have a hierarchical organisational structure, or *holarchy*. In systems such as these, there are many nested sub-organisations of agents[54]. These agents are autonomous, and may decide to join or leave holons, or come together to form new holons, dependent on their goals.

As part of each holon, a *holon head*, an agent that represents the holon to the external environment, is responsible for coordinating actions inside the holon[54]. Agents in holons can be viewed as forming a super-agent, abstracted up a level, with super-agents now communicating with each other through one holon head inside each grouping[228]. Viewing the organisation like this, we can see how the view of the system used by each individual agent can be restricted in size by the holon boundary. This allows an accompanying reduction in knowledge, and communications overhead required to function, and so computations become more feasible in larger systems as compared to agent structures without interaction boundaries.

In practice, holonic concepts provide us with a way to model multi-agent systems and to design them to encapsulate the necessary functionality and low coupling amongst agents[229]. Framed in terms of the task allocation problem we look to tackle, a holon head would be an agent receiving a set of tasks from outside the holon, which would then be allocated to other agents inside the holon. In turn, each agent receiving tasks inside of the holon could be acting as a holon head for another holon (Figure 6.1). This gives the system a *hierarchical task allocation* capability, which we utilise in our work in Chapter 11. With a stream of tasks incoming to the system that change with time however, the most efficient holonic structure itself may change.

6.6 Summary

In covering self-organisation we have looked at how structures and organisations can be formed and understood. To do so, agents need to be able to adapt as they gain new knowledge about their view of the system. With the application of learning algorithms, the required adaptability can be introduced, with structure and roles within the system developing naturally as the result of the task and resource allocation strategies of each agent. This topic will be part of our work in Chapters 11 and 12.

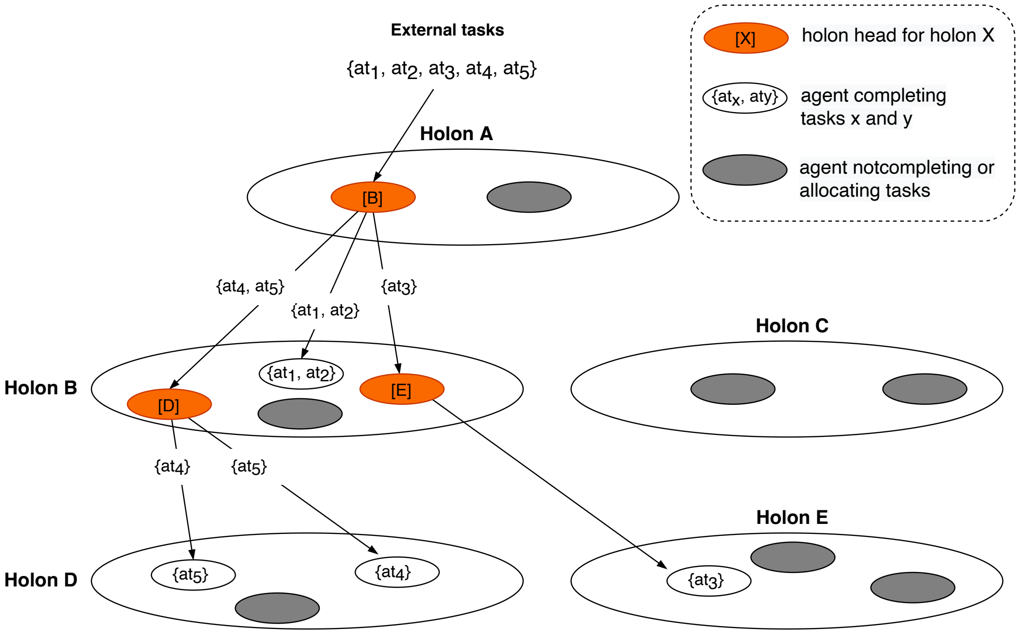


Figure 6.1: Overview of the holarchy structure of task allocation in a multi-agent system. *External tasks* $\{at_1, at_2, at_3, at_4, at_5\}$ *arrive in the system. The head for holon B allocates tasks to agents in that holon. Tasks* $\{at_1, at_2\}$ *are completed by an agent in that holon. The remaining tasks* $\{at_3\}$ *and* $\{at_4, at_5\}$ *are allocated by holon heads for holon D and E respectively, reallocated to agents within those holons, and completed.*

Chapter 7

Wireless sensor networks

In this chapter we look at wireless sensor networks in order to understand the challenges our algorithms must overcome in practical MAS applications.

7.1 Introduction

Wireless sensor networks (WSNs)

Wireless sensor networks (WSNs) are collections of independent nodes connected through wireless transmission, often found in situations where a geographical area requires monitoring using sensors[9], [12]. Due to the autonomy required of the nodes, they are well suited to MAS applications[230]. WSN provide concrete scenarios in a number of fields with strong commercial, or scientific uses, and where solving task and resource allocation problems would bring significant value. As such we will use these systems to explore how learning algorithms and multi-agent strategies can be used as solutions to real-world problems.

Networks are often deployed in areas that are difficult to access, such as remote locations, ocean-based monitoring, or contaminated regions. These environmental WSNs in particular are characterised by intermittent connectivity between nodes, with harsh environmental conditions that lead to degradation of the network, so that systems must adapt in response to these changes to maintain functionality[231]. The energy sources for nodes are commonly non-replaceable batteries, and the dispersal of nodes ad-hoc, making a high level of autonomous communication, coordination, and power consumption management essential.

WSNs can consist of static sensor deployments, or as mobile agents, adding to the need for adaptation in the network[232], [233]. Sensors are usually small and inexpensive, capable of utilising limited compute and storage resources. They have one or multiple sensing devices to take measurements from the environment, ranging through chemical, optical, thermal, biological, and radioactivity detection. This collected data is often transmitted to a base station and retransmitted to be stored remotely and analysed.

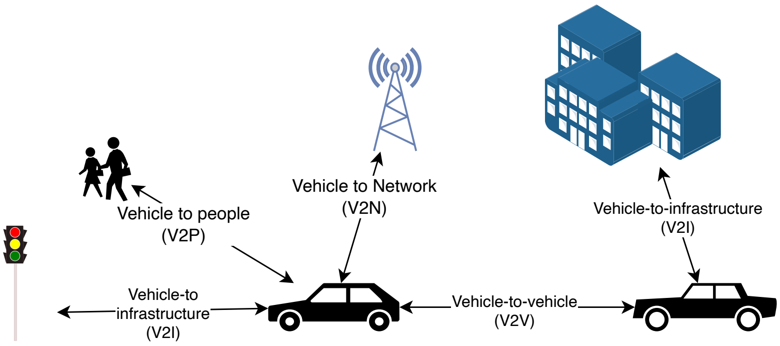


Figure 7.1: V2X agent coordination. *Agents representing vehicles can communicate and coordinate with; other vehicles to avoid congestion (V2V), infrastructure such as traffic lights (V2I), people in order to avoid accidents and arranging pick-ups (V2P), or network transmitters to integrate with automated city planning (V2N). As these interactions are between vehicles and a heterogeneous group of other agents, we use the catch-all term vehicle-to-everything (V2X).*

Chapter structure

In the following section, Section 7.2, we look at practical examples of WSN systems, which allows us to abstract out common objectives for such systems in Section 7.3. We look at general algorithms for such systems in Section 7.4 and learning algorithms in particular in Section 7.5. These approaches and limitations of such leads us to specify key criteria for our algorithms in tackling task and resource allocation problems in systems such as WSN in Section 7.6.

7.2 Examples of WSN systems

Vehicle-to-everything networks

As the concept of smart cities has developed, so has the desire for more intelligent traffic management. As such, developing algorithms to utilise vehicle-to-everything (V2X) communications has grown in relevance. V2X encompasses a set of categories of different agent-to-agent communications, where vehicles might talk to infrastructure such as lights and buildings, to devices such as mobile phones carried by pedestrians, or to other vehicles to orchestrate their actions (See Figure 7.1). Not only is the successful development of V2X systems of benefit to safety on the roads[234], but it also allows for more efficient use of fuel for vehicles travelling[235], and better management of traffic to reduce congestion[64].

Unmanned aerial vehicles (UAV)

UAVs, or drones, have found uses in a number of areas, particularly in those that can benefit from autonomous collection of data across a large area[236]. In agriculture[6], they are used to check the health of large areas of farmland, and tackle problems early to stop them spreading and destroying crops. In the defence industry swarms of UAVs are used to share logistics information and provide support in conflict areas[237]. They are also used for environmental data collection, such as for flood monitoring[238], or monitoring the state of forest habitats[239]. UAVs can provide an option for large-area search and rescue missions in remote areas with difficult to reach terrain[240], or ocean rescue[241]. These uses typically require ad-hoc network formation amongst the UAV to allow them to coordinate or relay information through to a base station, resource management[58] to ensure they stay operative in the air, and task allocation to accomplish the required goals.

Environmental sensor networks

WSNs have many environmental monitoring applications and ongoing research in areas such as oceanographic measurement[8], [11], [242], radioactive contamination[243], water quality[244], flood risk levels[245], volcanic activity[246], agricultural soil[247], as well as in military uses[248]. More recently, the availability and lower cost of low-power wireless transmitters[249], solar-harvesting components[250], and micro-electro-mechanical systems[251] has allowed for larger deployment sizes, expanding their real-world applications and opening up new areas for research[10], [252].

Radioactive contamination

Radioactive leaks and contamination require minimising human interaction with affected areas, vital for health protection. However, there is still a need to monitor these environments to judge the risk. By utilising WSN, at-a-distance deployments can be made, with the networks being formed on an ad-hoc basis. In addition, the damaging effects of radiation require robust networks that can adapt to the loss or damage to nodes. These use cases apply to on-premise deployments inside an at-risk facility[253], the monitoring of contamination leaks through water sources[254], or the coverage of remote, or otherwise inaccessible areas[243] that are impacted by radioactivity (See Figure 7.2).

Oceanographic monitoring

Monitoring marine environments has become a focus of recent years as part of ongoing research into the impact of pollution and climate change through the measurement of changes in water temperature, salinity, pH, oxygen levels, etc. Where traditional approaches to data collection such as a research vessels can be expensive and slow to collect data across large expanses of ocean, WSNs are hoped to be able to provide a much more granular, cheaper, and longer duration solution to data collection[11], [242]. These environments are harsh, and so sensor deployments must be resilient. Currents and waves mean motion or drift of equipment, salinity and turbulence can cause damage to components, and energy consumption must be optimised as well, as maintenance or replacement of components is difficult in these isolated and challenging environments.

Typical deployments range from sensor equipment attached to moored buoys, to underwater autonomous vehicles[255]. Examples can be found in monitoring near-shore lagoons for multiple factors such as temperature profile, pressure, currents, salinity, turbidity, as well as oxygen density, giving a detailed map of the waters and how they change over time[256]. Sensor networks have also been utilised to obtain a vertical profile of temperature in the Baltic Sea[257], to monitor light levels and temperatures around endangered coral reefs in Queensland, Australia[258], and discover the

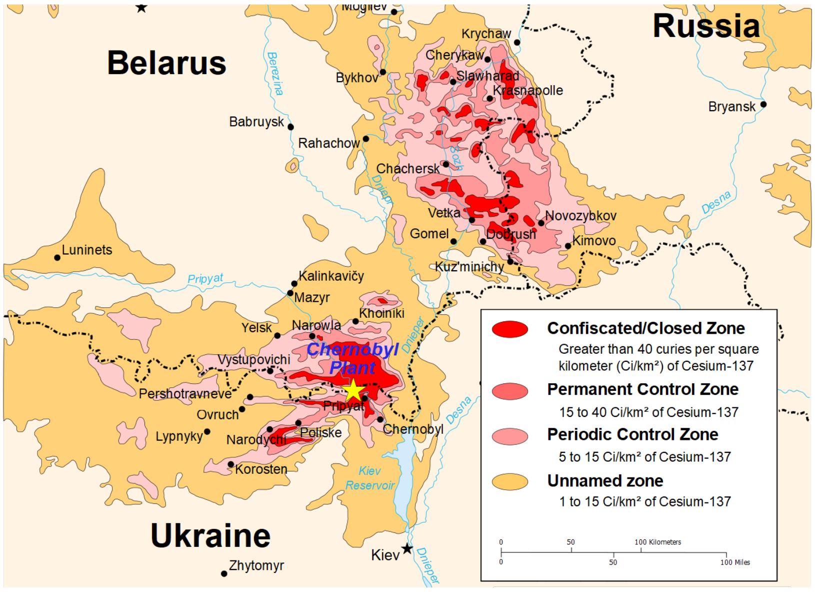


Figure 7.2: Map of radioactive contamination due to the Chernobyl disaster. *The map shows the concentration and distribution of radioactive contamination after the explosion of the Chernobyl nuclear power plant in Ukraine, 1996. (source: CIA Factbook, CC BY-SA 2.5, via Wikimedia Commons)*

optimal feeding, and so reduce pollution, in marine fish farms[259].

Solar harvesting Through greater efficiency of solar-panel energy harvesting[108] and reduction of the energy of wireless communications through acoustic signalling[260], these ocean-based WSNs are increasingly pragmatic to use in wide-scale deployments. With the natural dynamism of the marine environment, MARL techniques are also being applied to improve the resilience and adaptability of these networks[261].

Figure 7.3 shows some of the common components of marine WSN deployments as demonstrated by the European Unions' NeXOS project.

7.3 Objectives of a WSN system

The scenarios described above highlight five key requirements for a WSN, each of which present challenges in deploying and operating a WSN system.

Energy consumption Each node in a WSN network has limited power available, supplied by a battery. Depending on the environmental conditions, there may be some form of energy-capture component built into each node, such as solar-harvesting[250]. Batteries cannot be easily replaced in the remote or inhospitable locations that are often the focus of WSN, so minimising energy consumption is essential[264]. This can be done through the use of low-power components[265], [266], as well as applying energy-aware routing protocols[267].

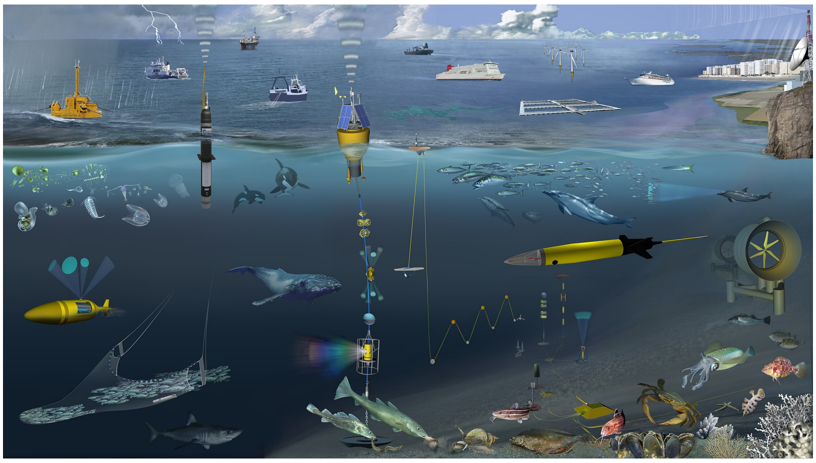


Figure 7.3: A diagram of oceanographic monitoring as envisaged by the NeXOS project. *The NeXOS project is a collaborative project funded by the European Commission 7th Framework Programme to develop cost-effective, innovative sensor networks for ocean monitoring and data collection[262], [263]. (source: Glynn Gorick, Eric Delory, Jay Pearlman, CC BY-SA 4.0, via Wikimedia Commons)*

Quality of measurement

A node taking a reading may have a faulty sensor, leading to variations in the recorded values and lack of reliability. Sensors may get more accurate readings using more energy or longer time scales, for example, as the sampling time of a temperature or radiation sensor is increased, the more accurate the reading becomes[268]. Therefore, nodes must trade off the quality of their acquisition of data with the restricted amount of resources available to them during their lifetime[269].

Sensor coverage

In many environmental situations sensors are distributed in an ad-hoc manner, meaning their distribution is initially unknown amongst the nodes. Sensors may also have occlusion problems due to the topography of the environment, or objects blocking connectivity or measurement[270]. Therefore, to initialise the system, the deployed nodes must find which other nodes to communicate with that will allow all the locations targeted by readings to be covered. They must also be resilient to temporary or permanent outages on the network that require re-routing connectivity to maintain this coverage.

Network resilience

Wireless sensor networks add substantial additional risks to reliability over standard networking. Nodes can run out of power, or components may fail, problems that are often exacerbated by harsh conditions. Environmental effects or obstacles may physically impact transmission or reception of signals. Loss of communication to a node is especially impactful as there are often multi-hop routes involved, which multiplies the risks[271]. To mitigate this problem, the WSN must be able to reconfigure its routing pathways to work around nodes that are no longer functioning so that other nodes required to take a sensor measurement can still be reached.

System lifetime Due to the effects of environmental degradation, power exhaustion, and connectivity loss, nodes in a network have a limited useful lifespan[272]. As nodes are lost, the system itself becomes degraded. Eventually it is unable to achieve its goals to a sufficient quality to be useful, defining the system's lifetime. To extend this lifetime as far as possible we try to reduce the wear on nodes, principally by ensuring that energy consumption is distributed throughout the system[273], [274].

To summarise, we require that WSNs; minimise their energy usage, distribute that energy usage to reduce the component wear on individual agents (to increase their working lifetime), provide measurements of sufficient quality that cover the required area, and adapt to network disruptions to maintain these properties.

7.4 Algorithms for WSN systems

Implementations of algorithms to WSNs can be found in a wide range of industries, from vehicle-to-vehicle communications to large-scale environmental monitoring. These systems commonly need to manage energy usage, maintain availability, distribute tasks effectively, and handle node communication failures. Decentralised algorithms are commonly used to meet these challenges, with hierarchical cluster formation or reinforcement learning techniques. There are challenges however in getting these algorithms to perform well where there are multiple objectives, where agents are mobile, or the connectivity between agents varies over the system's lifetime.

WSNs consist of sets of *nodes*, devices that connect to each other to form the network, with centralised or decentralised control. With centralisation, the controlling node has system-wide knowledge, using this to allocate measurement tasks to other nodes, orchestrate their communications, and handle recovery (see Figure 7.4). This approach does not scale well to large networks due to congestion and resource exhaustion on the central component. Additionally, in harsh environments this centralisation of control is not robust to damage or node loss. Although some adaptivity can be added to these systems to help them improve their performance in meeting their goals in complex systems, non-distributed learning algorithms suffer from the same limitations as non-learning WSN systems[275]. For these reasons, we focus on decentralised, autonomous methods.

With decentralised WSNs, nodes have limited knowledge of the system. Each node acts autonomously to some degree to orchestrate the functionality mentioned above. They are often organised into groups to decrease the cost of coordination without full centralisation. The basis of many of these decentralised techniques is hierarchical (see Chapter 6, Section 6.5), typically some form of clustering technique such as LEACH or similar algorithms[276]. Nodes are formed into subgroups with designated leaders that orchestrate the behaviours of each group of nodes and communicate with the central controller (see Figure 7.5).

This ability for nodes to work autonomously increases resilience but introduces new challenges. In working with local information only, nodes may take the best actions for their local environment, but which are less beneficial to a system's goals overall. In addition, system-wide coordination of node behaviours is difficult when communication is limited to small sets of neighbours[277]. Reinforcement learning has seen applications in these decentralised systems, often focused on using adaptive routing

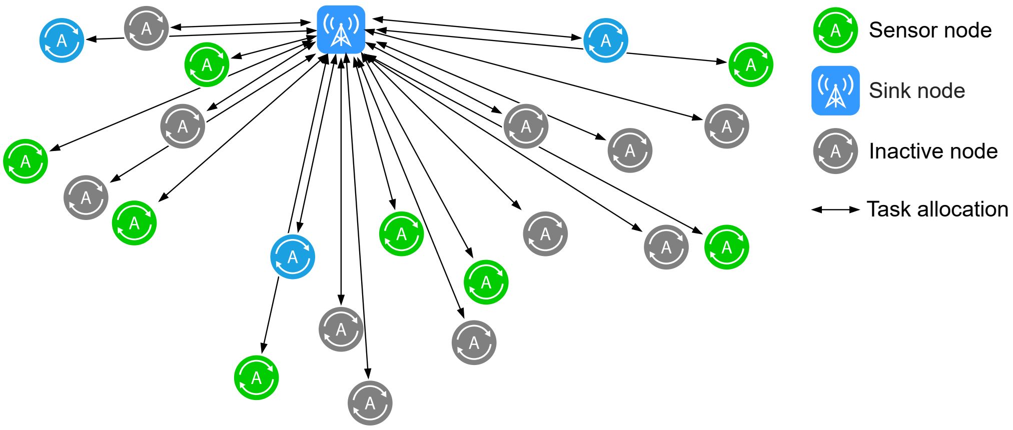


Figure 7.4: Centralised WSN configuration for task coverage of a grid. *In a centralised WSN, there is a coordinating agent that coordinates other agents in the system through direct communication and orchestration of their actions.*

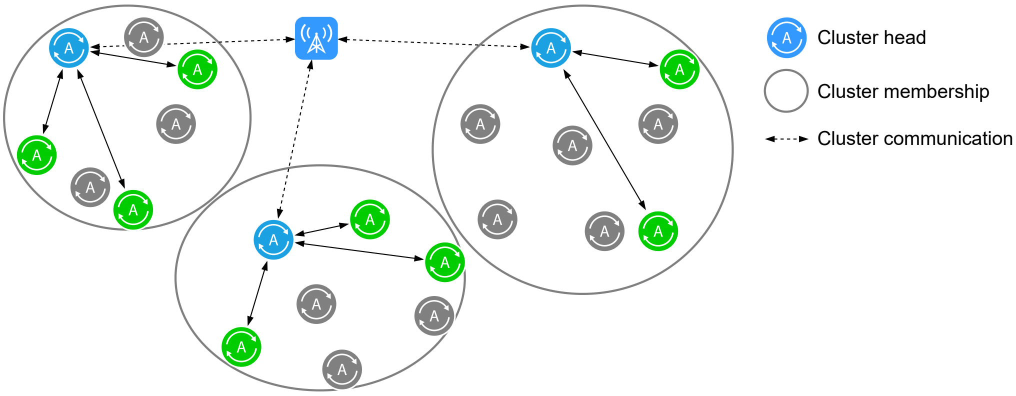


Figure 7.5: Decentralised WSN configuration for task coverage of a grid. *In a decentralised, clustered WSN, agents in the system split into groups. One agent in the group acts as a* cluster head*, an agent that coordinates the other agents in the group. These cluster head agents themselves are coordinate by another agent higher up the hierarchy.*

to optimise energy efficiency and system lifetime[278], [279], or to ensure sensor coverage is maintained despite the loss of nodes over time[280]. These solutions often look at single or complementary objective optimisation, and do not necessarily take account of the conflicting multiple objectives commonly found in WSN systems.

7.5 Learning strategies for WSN optimisation

The nature of WSN systems fit well with work on multi-agent systems. The best network connectivity, task allocation strategy, and assignment of resource by agents are unknown at system initialisation, and vary throughout the system's lifetime. As mentioned previously, reinforcement learning techniques are useful in learning these values as the system evolves[20]. These broadly have two main challenges; how algorithms can best learn to optimise across multiple system goals, and how agents share information and cooperate to connect their localised actions to the broader system goals.

Evolutionary algorithms The problem of optimising WSN systems across multiple objectives can be tackled through the use of multi-objective evolutionary algorithms[281], where the developed techniques have been integrated into algorithms for the WSN domain[282], [283]. In particular, *differential evolution (DE)* techniques have proven a simple and often effective way of optimising multi-agent systems[284]–[286]. These have been utilised in both multi-objective[287], and non-stationary systems[288], [289] such as WSNs[290].

There are factors that can make DE algorithms less applicable in real-world situations;

- *occlusion and obstruction*, nodes in a WSN may be obstructed by other objects or weather effects in an environment, making some actions less optimal or even possible. E.g, a transmission to another node is blocked by a rock;
- *bandwidth limitations*, nodes have limited transmission bandwidth and processing power, so its necessary to restrict the amount of information required to be exchanged between nodes to apply algorithms. The large quantities of data transfer to a centralised node needed to compute evolutions can become a significant bottleneck;
- *centralised coordination and reliability*, with increasing system size, and as nodes have a limited transmission range, data must be relayed through more and more intermediate nodes. In addition, connectivity to such nodes may be intermittent, disconnecting sections of the WSN;
- *heterogeneity*, nodes may be homogenous in specification at system initialisation, however, localised conditions, ad-hoc placement, and changing functionality over time, differentiates their capabilities and optimal actions over time. E.g, the degradation in transmission power as a node's battery wears down;
- *uniqueness due to location*, each node's optimal actions are often unique, such as having localised sensing conditions. E.g. a node in an ocean-based WSN may have its temperature sensor placed in a strong current, with large variations in subsequent readings;
- *mobility*, identical nodes in the same location may be mobile to different degrees and would need to take differing actions. E.g. a node attached to a stationary

buoy in an ocean environment does not need to spend as much time on actions to discover other nodes in the network as a node in the same location that is floating freely. A node's mobility may also change within its lifetime, for example with changing currents.

Distributed algorithms

These practical questions arising from WSN applications reinforce the reasoning behind the theoretical choices for our algorithms. As such, we look to distributed algorithms for application to WSN[291] with a focus on learning techniques[275] for better task scheduling and energy consumption such as cooperative Q-learning between nodes[292]. Due to the increased resources and computation required to coordinate amongst multiple agents and form cooperative behaviours, we also focus on decentralised, autonomous agents learning from localised knowledge only[293]. E.g. individual agents optimising energy usage within a small neighbourhood of nearby nodes to optimise the energy consumption of the system as a whole.

7.6 Key requirements of a WSN optimising algorithm

The systems we have discussed guides the choices of some desired behaviours of WSN algorithms;

1. *optimisation of multiple objectives*, the algorithm must be able to balance learning to take actions that benefit more than one objective so that it can optimise across energy use, task quality, etc[281], [283], [294];

2. *decentralisation of coordination*, with limited resources for transmission and computation, nodes in WSN should minimise the explicit coordination of agents. In harsh environments, and with limited possibilities for manual maintenance, centralised coordination or knowledge becomes both difficult and a danger to system robustness;

3. *localisation of agent knowledge*, nodes have limits on transmission range and bandwidth, which discourages the increasing resource costs of relaying transmissions, and the risk of network routes containing many hops. To avoid these issues, nodes should be restricted to information available nearby[293]. For this reason, we focus on using only the knowledge in an agent's local neighbourhood in our algorithm, and use this to guide coordinated behaviours in the system overall;

4. *allocation of node resources*, nodes in a WSN have limited resources with which to complete their tasks. Each node's preference of how its available resources are allocated amongst the tasks allocated to it will have an effect on how well it completes those tasks. An algorithm should be able to learn the allocation of resources at each node that optimises overall system utility.

7.7 Summary

In this chapter we looked at how wireless sensor networks can be used, and the difficulty of developing successful algorithms due to the challenges encountered in real-world deployments. We used these justifications to develop the high-level theoretical framework we will build on in Part II. We look to improve on previous work by developing algorithms that can meet the key objectives and requirements described, while still giving flexibility in the optimisation balance between the multiple system objectives of minimising energy consumption, while maximising system lifetime, measurement quality, and sensor coverage. The problems identified in operating a WSN can be defined as a multi-objective learning problem in a distributed MAS. From our initial task and resource allocation solutions in Chapter 9 and 10, we combine and extend these to tackle this problem in Chapters 11 and 12.

Part II

Research contributions

Chapter 8

Distributed task allocation systems (DTAS)

This chapter will provide the notation and concepts required to model allocating tasks in multi-agent systems. We use this chapter's theoretical work as the basis for defining the task and resource allocation problems, and our solutions to them, in the chapters that follow.

8.1 Introduction

Informally, a *distributed task allocation system (DTAS)* is a MAS where a set of agents work together to perform a set of tasks[72]. The main objective of the system is to maximise the number of successfully completed tasks, and the overall system utility, through coordination amongst the agents. The agents can join and leave the system at any time throughout lifetime of the system, and may be heterogeneous, i.e. have differing capabilities in completing tasks or performing actions.

We define the task-allocation system in Section 8.2, and its state dynamics in Section 8.3. We formally lay out the actions agents can take in Section 8.4, and some useful functions on states and actions in Section 8.5.

We will use the notation and model developed in this chapter; to define the task and resource allocation problems in Chapters 9 and 10; to develop functions for the quality, system utility, and optimality of allocations; as well as to specify our algorithms formally in Chapter 9, Section 9.5, Chapter 10, Section 10.4, and Chapter 11, Section 11.3.

8.2 Defining the DTAS

Given a set of tasks in a DTAS, there may be overlap in the results they produce, causing duplication, and making them less valuable to the system. Some tasks may produce unique and important results, making them more valuable. In other words, the value of the outcome of an individual member of a set of tasks can be dependent on the results of the other tasks in the set. Therefore, a task's true value to the system may only be known once all the interdependent tasks have been completed.

Atomic and composite tasks To allow for this we define two distinct types of tasks; *atomic tasks*, which are the tasks executed by agents; and *composite tasks*, which are formed as sets of these atomic tasks. As an atomic task comprising a composite task may be dependent on the output of other atomic tasks of the composite task, its value must be calculated through the value of the composite task as a whole.

System definition We define a DTAS as a set of agents working together to perform a set of composite tasks, formed by atomic tasks that can be executed by individual agents. Each agent has some *capabilities* to perform atomic tasks and is also able to coordinate and oversee the execution of composite tasks. Agents can perform *actions* that will update the system state. We use $\mathcal{G}$, to denote the set of all possible agents, $\mathcal{CT}$ for all possible composite tasks, $\mathcal{AT}$ for all possible atomic tasks, and $\mathcal{A}$ for all possible agent actions.

Task types Each atomic task $at \in \mathcal{AT}$ is typed by one of the set of all *atomic task types* $\mathcal{AP}$, defining what repeatable activity has to be performed to complete that task. The function $type_a \colon \mathcal{AT} \rightarrow \mathcal{AP}$ maps atomic tasks to their respective task types. Similarly, each composite task $ct \in \mathcal{CT}$ is typed by one of the set of all *composite task types*, $\mathcal{CP}$, defined by the types of all the atomic tasks that compose that composite task. Therefore, a composite task can be mapped to the types of its respective atomic tasks, $type_c \colon \mathcal{CT} \rightarrow \mathcal{CP}$, where $type_c(\{at_1, .., at_n\}) = \{type_a(at_1), .., type_a(at_n)\}$.

Neighbourhood We assume that for each agent in a DTAS there is a subset of other agents in the DTAS which it can send messages to and receive messages from, called the *neighbourhood* of that agent. We assume that the neighbourhood of an agent can change over time but that it is potentially limited in size due to the agent's software and hardware constraints. Due to these constraints, it may not be feasible for all agents in the DTAS to be part of an agent's neighbourhood, therefore changing the neighbourhood can have a cost due to the agent losing access to other, previously accessible agents.

Similarly, we assume that there is a limited number of other agents in the DTAS that an agent can store information on, its *knowledge*, which also may change over time. With limits to the amount of knowledge an agent can retain at any one time, changing knowledge can mean that the agent loses awareness of other agents in the system it could possibly bring into its neighbourhood. **Knowledge**

Definition 8.2.1 (*Distributed Task Allocation System*)**.** A distributed task-allocation system (DTAS) is defined by a tuple $\langle AT, CT, A, G\rangle$ where:

- $AT \in \mathcal{AT}$ is the set of atomic tasks in the system at any point during its existence, where each task can be performed by a single agent;
- $CT \in \mathcal{CT}$ is a set of composite tasks in the system at any point during its existence, where each composite task is formed from a set of atomic tasks;
- $A \in \mathcal{A}$ is the set of actions that agents G can perform;
- $G \in \mathcal{G}$ is the set of agents that can operate in the system at any point during its existence. Each agent $g \in G$ is defined by a tuple $\langle id, c, r, \delta_k, \delta_n\rangle$, where;
  - id is a unique identifier for the agent;
  - $c \subseteq AP$ is the agent capabilities (i.e. the types of atomic task that the agent can perform);
  - $r \subseteq CP$ is the agent responsibilities (i.e. the types of composite task that the agent can oversee);
  - $\delta_n \in \mathbb{N}$, is the agent's communication constraint;
  - $\delta_k \in \mathbb{N}$, is the agent's memory constraint.

Given an agent g, we denote by $c(g), r(g), \delta_n(g), \delta_k(g)$ the capabilities, responsibilities, communication, and memory constraints of that agent, respectively.

For all atomic tasks, we assume there is at least one agent capable of performing it, $\forall at \in AT\ \exists\, g \in G : type_a(at) \in c(g)$. Similarly, for all composite tasks in the system there is at least one agent responsible for overseeing it, $\forall ct \in CT\ \exists\, g \in G : type_c(ct) \in r(g)$.

We define the *atomic task completion* function $complete\colon 2^{\mathcal{AT}} \rightarrow \mathbb{B}$, which returns 1 if all the atomic tasks in the given set AT have been executed by agents, and the results returned to the agent whose composite task contains the atomic task, and 0 otherwise. **Task completion**

Example 8.2.1 (*Atomic and composite tasks in a WSN*). The agents in a marine-based WSN system are equipped with sensors that can complete tasks to measure salinity, oxygen levels, and water acidity, so $AP = \{ap_{sal}, ap_{oxy}, ap_{ph}\}$. Each agent's capabilities may be a subset of these atomic task-types depending on which sensors they have, and whether they are functional. For instance, if an agent g only has working sensors to measure salinity and oxygen levels then $c(g) = \{ap_{sal}, ap_{oxy}\}$. Some agents receive composite tasks from outside the system, requests for samples of combinations of these measurements, e.g. $ct = \{at_{sal}, at_{oxy}\}$. These agents then decompose these composite tasks into atomic tasks and allocate them to other agents to complete. Assume one of the agents has a composite task $ct = \{at_{ph1}, at_{ph2}, at_{ph3}\}$, where the measurement location of at_{ph2} is nearby to at_{ph1}, but that of at_{ph3} is far away from both at_{ph1} and at_{ph2}. In this case, at_{ph2} may replicate data already returned from at_{ph1}, and so at_{ph2} is less valuable to the composite task. However, at_{ph3} produces unique results and therefore has an increased value to the composite task.

For the following sections and much of the thesis we will refer to our defined system as 'the DTAS' to allow for brevity as we do not need to parameterise our approach by which DTAS is under consideration. However, the specifications and contributions apply equally to any DTAS.

8.3 Dynamics of the DTAS

Composite tasks arrive in the system with constant or slowly varying frequency distribution[1]. The DTAS is capable of processing these tasks in the following way:

1. a composite task (of a defined composite task type) arrives in the system;
2. the composite task is allocated to an agent that can be responsible for tasks of that type;
3. the agent decomposes the composite task into atomic tasks;
4. the agent allocates these atomic tasks to other agents, or itself, if it has the capability;
5. the agents that have been allocated atomic tasks complete them and return the results of each atomic task to the agent that allocated them that task;
6. once all the atomic tasks have been completed the composite task is complete.

Agent state To be able to allocate atomic tasks, agents need to not only to be aware of the other agents in the system and their capabilities to execute tasks, but also to have communication links with those agents. Therefore, we define the current state of an agent by its knowledge, and neighbourhood.

Definition 8.3.1 (*Agent State*). Given an agent $g = \langle id, c, r, \delta_k, \delta_n \rangle$, we define its state as a tuple $\langle K, N \rangle$, where:

[1] This assumption is necessary to allow our algorithms, described in later chapters, to learn these distribution patterns. With no variation, the DTAS can be pre-designed and non-adaptive. With too much variation, the incoming task pattern cannot be sufficiently predictable for the system to adapt accordingly.

- $K \subseteq G$ is the knowledge of the agent[2];
- $N \subset K$ is the neighbourhood of the agent.

Given an agent g we denote by $K(g)$ and $N(g)$, its knowledge and neighbourhood. Note that, $\forall g \in G$, $|K(g)| \leq \delta_k(g)$ and $|N(g)| \leq \delta_n(g)$.

We denote by G_S the set formed by the states of each individual agent in the system, and the state of a particular agent g as $G_S(g)$.

Parent and child agents

Given a system of all possible agents $\mathcal{G}$, then we define *external agents* $\mathcal{E}$ as all the agents outside this system that can allocate the system composite tasks. Composite tasks arrive in the system from external agents to *parent agents* $pg \in G$, where parent agent is a role that an agent plays with regard to a composite task by overseeing its completion. A parent agent can allocate atomic tasks that compose a composite task to other agents, or execute them itself if it is capable of doing so. An agent which has been allocated an atomic task to complete assumes the role of a *child agent* for that task, $cg \in G$.

Actions

Agents in the system can take *actions*, $a \in A$, either by executing those actions themselves, or by allocating certain types of actions to other agents to complete. E.g. a car in a V2X system may take an action to update its knowledge of nearby congestion, which involves allocating one or more actions to other cars for them to send back information on traffic near them.

Allocations

The state of *allocations* of tasks and actions within a system are defined by three tuples, the *composite allocation*, the *atomic allocation*, and the *action allocation*. Each element of a set of allocations is an *assignment*, a tuple of a task or action, an agent g that it will be allocated to, and $\widehat{g}$, the agent allocating the task or action to g.

Definition 8.3.2 (*Composite allocation*)**.** The *composite allocation* is the set of composite tasks, CT, present in the system, sent by external agents, E, and allocated to agents, G, responsible for their completion, $\mathcal{CL} = 2^{(\mathcal{CT} \times \mathcal{G} \times \mathcal{E})}$, where each assignment $\langle ct, g, e \rangle$ represents the allocation of a composite task ct, to a system agent g, by an external agent e.

Definition 8.3.3 (*Atomic allocation*)**.** The *atomic allocation* is the set of all atomic tasks AT that are present in the system, and allocated to agents in G by other agents (or themselves), to complete, $\mathcal{AL} = 2^{(\mathcal{AT} \times \mathcal{G} \times \mathcal{G})}$, where each assignment $\langle at, g, \widehat{g} \rangle$ represents the allocation of an atomic task at, to an agent g, by an agent $\widehat{g}$.

Definition 8.3.4 (*Action allocation*)**.** The *action allocation* is the set of all actions present in the system, A, that have been allocated to agents in G by other agents, to enact, $\mathcal{ACT} = 2^{(\mathcal{A} \times \mathcal{G} \times \mathcal{G})}$, where each assignment $\langle a, g, \widehat{g} \rangle$ represents the allocation of an action a, to an agent g, by an agent $\widehat{g}$.

[2] This definition of knowledge as agents that an agent knows about, and has retained information on, was chosen as the most basic form of knowledge that could be exchanged, while ensuring our algorithms demonstrated the behaviour of agents learning better optimisations of their performance through the use of shared knowledge (therefore keeping the scope of our work focused). The exchange of more complex types of knowledge was deferred to future work (See Chapter 13, Section 13.4).

System state The system can be one of all possible states $\mathcal{S}$. Agents can take actions that will transition the system between states, each action performed incrementing a *system logical time counter*, $\phi \in \mathbb{N}_0$. Over the systems' lifetime it will move through a set of such states $S \subseteq \mathcal{S}$.

Definition 8.3.5 (*System State*)**.** We define the *system state* as a tuple, $s = \langle G_S, CL, AL, ACT, \phi \rangle$ where:

- G_S is the set of states of all agents in the system;
- CL is the set of composite allocations in the system;
- AL is the set of atomic allocations in the system;
- ACT is the set of actions allocated in the system;
- ϕ is the system logical time counter.

Selecting tasks from allocations We define the mapping $atomics\colon \mathcal{AL} \to 2^{\mathcal{AT}}$ as the set of all atomic tasks included in an atomic allocation, such that:

$$atomics(AL) = \{at : \langle at, g, \widehat{g} \rangle \in AL\} \tag{8.1}$$

Similarly, for composite tasks included in a composite allocation $composites\colon \mathcal{CL} \to 2^{\mathcal{CT}}$:

$$composites(CL) = \{ct : \langle ct, g, e \rangle \in CL\} \tag{8.2}$$

Those atomic tasks in an atomic allocation that have been allocated to an agent g are given by its *concurrent atomic allocations*, $concurrent\colon \mathcal{AL} \times \mathcal{G} \to 2^{\mathcal{AT}}$ where:

$$concurrent(AL, g) = \{at : \langle at, g, \widehat{g} \rangle \in AL\} \tag{8.3}$$

The atomic tasks that are available to an agent g to allocate to other agents as a result of the decomposition of the composite tasks allocated to it are given by, $allocatable\colon \mathcal{CL} \times \mathcal{G} \to 2^{\mathcal{AT}}$, where:

$$allocatable(CL, g) = \{at : at \in ct, \langle ct, g, e \rangle \in CL\} \tag{8.4}$$

8.4 Agent actions

The state of a DTAS changes as a result of external agents assigning sets of tasks to the system, and the actions that are executed by the system's agents. We define below the possible actions $\mathcal{A}$ in the system, along with their operational semantics. These actions are; the assignment, allocation, and execution of tasks; information request, provision, and removal; and the addition and removal of agents from a neighbourhood.

Assignment action The ASSIGN action takes a composite task ct from an external agent e, and allocates it to an agent within the system, $g \in_R G$, an arbitrarily selected agent responsible for that composite task type[3]:

$$\frac{\begin{array}{l} ASSIGN(e, ct) \wedge e \in E \\ \qquad\qquad\qquad \wedge\ \exists g \in_R G : type_c(ct) \in r(g) \end{array}}{\langle G_S(g), CL, AL, ACT, \phi \rangle \to \langle G_S(g), CL \cup \{\langle ct, g, e \rangle\}, AL, ACT, \phi + 1 \rangle} \tag{8.5}$$

[3] Using the notation $x \in_R S$ to represent the arbitrary selection of a member x from a set S[295].

Allocation action An agent g performing an *allocation action* allocates an atomic task at to one of its neighbourhood agents, n. The atomic task must be one that is currently allocated to the agent g by another agent $\widehat{g}$, or as part of a composite task allocated to the agent g by an external agent e:

$$\frac{\begin{array}{l} ALLOC(g, at, n) \wedge g \in G \\ \qquad\qquad\qquad \wedge\ at \in AT \\ \qquad\qquad\qquad \wedge\ n \in N(g) \\ \qquad\qquad\qquad \wedge\ \{\exists \langle ct, g, e \rangle \in CL : at \in ct \ \vee\ \exists \langle at, g, \widehat{g} \rangle \in AL\} \end{array}}{\langle G_S(g), CL, AL, ACT, \phi \rangle \longrightarrow \langle G_S(g), CL, AL \cup \{\langle at, n, g \rangle\}, ACT, \phi + 1 \rangle} \tag{8.6}$$

We also define a restricted version of this action which can only be performed by parent agents allocating atomic tasks belonging to one of their composite tasks, and where, once allocated to an agent, that agent cannot then reallocate the task to other agents. We use this SINGLEALLOC action in developing our work in Chapter 9 and Chapter 10, before introducing the full solution, including the ability to reallocate tasks using the ALLOC action, in Chapter 11.

$$\frac{\begin{array}{l} SINGLEALLOC(g, at, n) \wedge g \in G \\ \qquad\qquad\qquad\qquad\qquad \wedge\ at \in AT \\ \qquad\qquad\qquad\qquad\qquad \wedge\ n \in N(g) \\ \qquad\qquad\qquad\qquad\qquad \wedge\ \{\exists \langle ct, g, e \rangle \in CL : at \in ct\} \end{array}}{\langle G_S(g), CL, AL, ACT, \phi \rangle \longrightarrow \langle G_S(g), CL, AL \cup \{\langle at, n, g \rangle\}, ACT, \phi + 1 \rangle} \tag{8.7}$$

Execution action An agent g can perform an *execution action* when it has been allocated an atomic task, at and has that capability. The atomic task can be one allocated by another agent $\widehat{g}$, or as part of an agent's composite task:

$$\frac{\begin{array}{l} EXEC(g, at) \wedge g \in G \\ \qquad\qquad\quad \wedge\ at \in AT \\ \qquad\qquad\quad \wedge\ type_a(at) \in c(g) \\ \qquad\qquad\quad \wedge\ \{\exists \langle ct, g, e \rangle \in CL : at \in ct \ \vee\ \exists \langle at, g, \widehat{g} \rangle \in AL\} \end{array}}{\langle G_S(g), CL, AL, ACT, \phi \rangle \longrightarrow \langle G_S(g), CL, AL \setminus \{\langle at, g, \widehat{g} \rangle\}, ACT, \phi + 1 \rangle} \tag{8.8}$$

Request information action An agent g performs an *request information action*, by allocating a PROVIDE_INFO action to an agent in its neighbourhood, n, to provide information:

$$\frac{\begin{array}{l} INFO(g, n) \wedge g \in G \\ \qquad\qquad\quad \wedge\ n \in N(g) \end{array}}{\langle G_S(g), CL, AL, ACT, \phi \rangle \longrightarrow \langle G_S(g), CL, AL, ACT \cup \{\langle PROVIDE\_INFO, n, g \rangle\}, \phi + 1 \rangle} \tag{8.9}$$

Provide information action An agent g performs a *provide information action* by sending information about an agent k, arbitrarily selected from its knowledge[4], to

[4] We use a simple form of knowledge here to contain the scope of our work, however this is easily expandable to cover the exchange of more complex types of knowledge (See Chapter 13, Section 13.4).

the agent $\widehat{g}$, that allocated it the respective PROVIDE_INFO action. The agent $\widehat{g}$ then adds k to its knowledge, as long as it has not reached its knowledge constraint limit:

$$\frac{\begin{aligned} PROVIDE\_INFO(g, \widehat{g}) &\land g \in G \\ &\land \widehat{g} \in N(g) \\ &\land \exists k \in_R K(g) \\ &\land \langle \texttt{provide\_info}, g, \widehat{g} \rangle \in ACT \\ &\land |K(\widehat{g})| < \delta_n(g) \end{aligned}}{\langle G_S(g), CL, AL, ACT, \phi \rangle \longrightarrow \langle G_S(g), CL, AL, ACT \setminus \{\langle PROVIDE\_INFO, n, g \rangle\}, \phi + 1 \rangle} \tag{8.10}$$

where $G_S(g) = \{\langle K(g'), N(g') \rangle | \forall g' \in (G_S(g) \setminus \{\widehat{g}\}\} \cup \{\langle K(\widehat{g}) \cup \{k\}, N(\widehat{g}), \rangle\}$

Remove information action An agent g can take a *remove information action* by removing an agent k from its knowledge as long as that agent is not in its neighbourhood:

$$\frac{\begin{aligned} REMOVE\_INFO(g, k) &\land g \in G \\ &\land k \in K(g) \\ &\land k \notin N(g) \end{aligned}}{\langle G_S(g), CL, AL, ACT, \phi \rangle \longrightarrow \langle G_S(g), CL, AL, ACT, \phi + 1 \rangle} \tag{8.11}$$

where $G'_S = \{\langle K(g'), N(g') \rangle | \forall g' \in (G_S(g) \setminus \{g\}\} \cup \{K(g) \setminus \{k\}, N(g)\}$

Link action An agent g can add an agent k in its knowledge into its neighbourhood, as long as it is within its neighbourhood constraint limit, by taking a *link action*:

$$\frac{\begin{aligned} LINK(g, k) &\land g \in G \\ &\land k \in K(g) \\ &\land |N(g)| < \delta_n(g) \end{aligned}}{\langle G_S(g), CL, AL, ACT, \phi \rangle \longrightarrow \langle G_S(g), CL, AL, ACT, \phi + 1 \rangle} \tag{8.12}$$

where $G_S(g) = \{\langle K(g'), N(g') \rangle | \forall g' \in (G_S(g) \setminus \{g\}\} \cup \{\langle K(g), N(g) \cup \{k\} \rangle\}$

Remove link action An agent g can remove an agent n from its neighbourhood by taking a *remove link action*:

$$\frac{\begin{aligned} REMOVE\_LINK(g, n) &\land g \in G \\ &\land n \in N(g) \end{aligned}}{\langle G_S(g), CL, AL, ACT, \phi \rangle \longrightarrow \langle G_S(g), CL, AL, ACT, \phi + 1 \rangle} \tag{8.13}$$

where $G'_S = \{\langle K(g'), N(g') \rangle | \forall g' \in (G_S(g) \setminus \{g\}\} \cup \{K(g), N(g) \setminus \{n\}\}$

8.5 Grouping and filtering actions

Action categories

In out work in Chapter 9 we will use the outcomes of an agent's past actions to adapt the probability of it taking certain groups of actions in the future. To simplify this, we define the *action-category* of an action as:

$$category(a) = \begin{cases} ASSIGN & \text{if } \exists e \in \mathcal{E}, ct \in \mathcal{CT} : \; a = ASSIGN(e, ct) \\ ALLOC & \text{if } \exists g, n \in \mathcal{G}, at \in \mathcal{AT} : \; a = ALLOC(g, at, n) \\ SINGLEALLOC & \text{if } \exists g, n \in \mathcal{G}, at \in \mathcal{AT} : \; a = SINGLEALLOC(g, at, n) \\ EXEC & \text{if } \exists g, at \in \mathcal{AT} : \; a = EXEC(g, at) \\ INFO & \text{if } \exists g, n \in \mathcal{G} : \; a = INFO(g, n) \\ PROVIDE\_INFO & \text{if } \exists g, \widehat{g} \in \mathcal{G} : \; a = PROVIDE\_INFO(g, \widehat{g}) \\ REMOVE\_INFO & \text{if } \exists g, k \in \mathcal{G} : \; a = REMOVE\_INFO(g, k) \\ LINK & \text{if } \exists g, k \in \mathcal{G} : \; a = LINK(g, k) \\ REMOVE\_LINK & \text{if } \exists g, n \in \mathcal{G} : \; a = REMOVE\_LINK(g, n) \end{cases} \tag{8.14}$$

Available actions

We define the *available actions* of an agent g in a state $s = \langle G_S(g), CL, AL, ACT, \phi \rangle$, where $G_S(g) = \langle K, N \rangle$, as $actions \colon \mathcal{G} \times \mathcal{S} \to 2^{\mathcal{A}}$, such that $actions(g, s)$ returns the set of all the actions that the agent g can perform in its current state:

$$\begin{aligned} actions(g, s) = \; & \{ALLOC(g, at, n) : at \in concurrent(AL, g), n \in N\} \\ & \cup \{SINGLEALLOC(g, at, n) : at \in allocatable(CL, g), n \in N\} \\ & \cup \{EXEC(g, at) : type_a(at) \in c(g)\} \\ & \cup \{INFO(g, n) : n \in N\} \\ & \cup \{LINK(g, k) : k \in K, k \notin N\} \end{aligned} \tag{8.15}$$

Target actions

We define *target actions*, $targets \colon \mathcal{G} \times \mathcal{S} \times 2^{\mathcal{G}} \to 2^{\mathcal{A}}$, such that $targets(g, s, G)$ returns the set of all the available actions for the agent g in a state s, that interact with an agent in the set G:

$$\begin{aligned} targets(g, s, G) = & \{a \in actions(g, s) : a = ALLOC(g, at, n), n \in G\} \\ & \cup \{a \in actions(g, s) : a = SINGLEALLOC(g, at, n), n \in G\} \\ & \cup \{a \in actions(g, s) : a = INFO(g, n), n \in G\} \\ & \cup \{a \in actions(g, s) : a = LINK(g, k), k \in G\} \end{aligned} \tag{8.16}$$

Unknown states

There are states an agent could be in, but lacks the knowledge to access yet. We refer to these states as the *unknown states* of an agent g, $unknown \colon \mathcal{G} \times 2^{\mathcal{S}} \to 2^{\mathcal{S}}$. Given an agent's current state, $G_S(g) = \langle K, N \rangle$, and a set of states S, then for each $s \in S$ where the agent's state is $\langle K', N' \rangle$ then the set of states which does not contain the agent's current knowledge is given by:

$$unknown(g, S) = \{s : K \setminus K' \neq \emptyset\} \tag{8.17}$$

Example 8.5.1 (*Actions in a WSN*). An agent g_1 is part of an ocean-monitoring WSN containing agents $G = \{g_1, g_2, g_3, g_4\}$, where the current neighbourhood of g_1 is $N(g) = \{g_1, g_2\}$, and its knowledge is, $K(g) = \{g_1, g_2, g_3\}$. Agent g_1 receives a composite task $ct = \{at_{sal}, at_{oxy}\}$, where $type_a(at_{sal}) = ap_{sal}$ and $type_a(at_{oxy}) = ap_{oxy}$. Since agent g_1 has a working salinity measuring sensor, $ap_{sal} \in c(g)$, it can complete the task at_{sal} itself, and so performs action $EXEC(g_1, at_{sal})$. As it does not have a sensor to detect oxygen levels, it cannot complete tasks of that type, $ap_{oxy} \notin c(g)$, and so it allocates this task to an agent in its neighbourhood, g_2, through the action $ALLOC(g_1, at_{oxy}, g_2)$. In this case, due to its neighbourhood and knowledge, the available actions of g_1 in its current state s are:

$$\begin{aligned} actions(g_1, s) = \{&ALLOC(g_1, at_{oxy}, g_2), ALLOC(g_1, at_{sal}, g_2), \\ &EXEC(g_1, at_{sal}), INFO(g_1, g_2), LINK(g_1, g_3)\} \end{aligned}$$

and its current target actions for the set $\{g_2\}$:

$$targets(g_1, s, \{g_2\}) = \{ALLOC(g_1, at_{oxy}, g_2), INFO(g_1, g_2)\}$$

8.6 Summary

This chapter formally defined our DTAS, its dynamics, and how agents can act within the environment. In the next chapter we build on this to define the task allocation problem, and the algorithms we use to solve for it.

Chapter 9

Task allocation

In this chapter we look at the learning and optimisation of task allocation by agents in a dynamic environment. We develop new algorithms to tackle this problem and evaluate their performance in a range of simulated systems; a stable system to measure overall performance, i.e., how close our algorithm's system utility approaches the theoretical maximum achievable in the system; another to look at how our algorithms' exploration strategy performs; a volatile system to see how they adapt to heavily perturbed systems; and a large system to examine how well the algorithms' performance scales as the number of agents in the system increases (Contribution 1).

9.1 Introduction

In a DTAS we aim to have efficient task allocation in a dynamic multi-agent system while ensuring scalability, even as the number of tasks increases and the availability of agents changes. In this chapter we present four algorithms to solve this problem. Each algorithm is focused on tackling a different aspect of the problem, then they are combined to solve the problem as a whole. These algorithms enable each agent to improve their task allocation strategy through reinforcement learning, while changing how much they explore the system in response to how optimal they believe their current strategy is, given their experience.

Chapter structure The structure of this chapter is separable into four main areas; the task allocation problem itself, the techniques we use to adapt Q-learning for non-stationary environments, the algorithms we develop using these techniques to solve the task allocation problem, and their evaluation.

In the next section, Section 9.2, we describe our strategy for tackling Q-learning in a non-stationary environment. Section 9.3 introduces the task allocation problem more formally, alongside the notation and equations we will use to describe task allocation and system utility within a DTAS. We define locally-optimal, and system-optimal allocations, concepts which we then use in extending Q-learning to non-stationary environments in the next section.

Section 9.4 focuses on Q-learning in non-stationary environments, and the problem of agents learning a policy (their understanding of the system's state and which actions to take in it) which must change in response to changes in the environment. An agent does not know how much the environment has changed, how those changes affect the usefulness of different parts of its policy, and therefore how quickly to adapt its learning.

In the previous chapter we defined an agent's internal state as $\langle K, N \rangle$, with system state $\langle G_S, CL, AL, ACT, \phi \rangle$, where an agent is learning to optimise its task allocation through ALLOC and EXEC actions, and its internal state through LINK and INFO actions. The optimisation of task allocations does not involve decisions that will affect the longer-term performance of the agent, and can be tackled through common reinforcement learning algorithms. However, the optimisation of its internal state does affect the future decisions and performance of the agent. How the agent predicts the impact of such decisions is detailed in Sections 9.4.3-9.4.5. The choice of the agent on whether to focus on this allocation optimisation, or on internal state optimisation, on each time step is part of the work of the RT-ARP algorithm.

We detail our proposed algorithms in Section 9.5. This is followed by evaluation of the algorithms' performance through simulating various systems in Section 9.6, with analysis in Section 9.7. Finally, we look how this work relates to a practical application in an ocean-based WSN in Section 9.8.

9.2 Learning in a non-stationary environment

Our strategy is to enhance Q-learning to tackle the problems resulting from non-stationary environments by enabling agents to make predictions of the possible allocation quality achievable in their current neighbourhood, and how that quality might compare to possible values in other neighbourhoods. We briefly introduce standard Q-learning techniques in Subsection 9.4.1 before moving on to how we adapt them to tackle the problems resulting from non-stationary environments in Subsections 9.4.2 - 9.4.5. These subsections look at three areas;

1. *the value of information in a neighbourhood*, how do we enable agents to use rewards information from their past actions to estimate the value of their neighbourhood? I.e. how much has an agent learned about a neighbourhood and the best actions to take within that neighbourhood? The agent can use this estimation to decide if it has enough information to predict whether taking actions (that change its neighbourhood) would be beneficial to its performance. The

agent can also decide which agents and actions it should retain information on when managing its neighbourhood and knowledge, while keeping within constraints (Subsection 9.4.2);

2. *the impact of actions on an agent's state*, understanding this helps an agent predict how the actions it could take would change its neighbourhood and knowledge, and whether the locally-optimal allocation of its tasks in this new state would be better than in its current one. This predictive ability helps agents more efficiently select actions that will move them towards parts of state-space where they will achieve better performance (Subsection 9.4.3);
3. *allocation quality metrics*, developing allocation quality metrics allows an agent to judge the performance of its current policy as compared to its previous ones. In order to predict whether taking actions that would change its neighbourhood or knowledge would be advantageous, the agent needs to predict how close it is to the locally-optimal allocation quality in its current neighbourhood, and how close this value is to the system-optimal allocation quality, given that both these values are not known to it (Subsection 9.4.4).

By combining the work from these areas, the agent can then determine whether it should take actions that it currently predicts to be non-optimal, or actions that will substantially change its neighbourhood or knowledge base, in order to explore the action-space and achieve better performance.

9.3 Task quality and the optimality of allocations

Permutations of atomic tasks

Given a set of atomic tasks AT, and a set of agents G, there are a number of different $permutations\colon 2^{\mathcal{AT}} \times 2^{\mathcal{G}} \to 2^{\mathcal{AL}}$ possible when allocating these tasks amongst agents:

$$permutations(AT, G) = \{AL \in \mathcal{AL} : \forall \langle at, g, \widehat{g} \rangle \in AL, at \in AT, g \in G, \widehat{g} \in G\} \quad (9.1)$$

Atomic task quality

An agent completes each of its allocated atomic tasks to a quality. The definition of this quality is system-specific, depending on the system's goals. E.g. in an environmental monitoring system, the quality of an atomic task to take a measurement might have a strong dependency on the reading's accuracy. Whereas, in a V2X system, a task to calculate a vehicle's location may well place more value on how quickly the task is performed.

For our DTAS system, we make the assumption that an agent which has been allocated multiple tasks must share its resources amongst those tasks until they are completed, and that this will reduce the quality of completion of those tasks, i.e. the atomic task quality decreases as the number of concurrent tasks an agent is performing, $|concurrent(AL, g)|$, increases (see Example 9.3.1).

Definition 9.3.1 (*Atomic task quality*). The *atomic task quality* for an agent completing an atomic task depends on the task's type, and the concurrent atomic allocations of the agent, $atomicql\colon \mathcal{AP} \times \mathbb{N}_0 \to \mathbb{R}$

Definition 9.3.2 (*Atomic allocation quality*). The *atomic allocation quality* of a set of atomic tasks AT, given an atomic allocation AL, is defined as $allocql\colon 2^{\mathcal{AT}} \times \mathcal{AL} \to \mathbb{R}$, where:

$$allocql(AT, AL) = \sum_{at \in AT, \exists \langle at, g, \widehat{g} \rangle \in AL} atomicql(type_a(at), |concurrent(AL, g)|) \qquad (9.2)$$

Example 9.3.1 (*The effect of concurrency on task quality*). An agent g in an ocean monitoring WSN is to be allocated an oxygen measurement task at_{oxy}, and a salinity measurement task at_{sal}. The quality of completing these tasks is mostly dependent on the precision of the results rather than other factors such as how quickly the tasks are completed. By dedicating more of its power to a measurement task, the agent can increase the precision of the results of the sensor reading.

Given allocations AL_1 and AL_2, where at_{oxy} and at_{sal} are allocated to g independently (i.e. at_{sal} was only allocated to g after at_{oxy} had completed), then g can allocate 100% of its available power to each task, obtaining a high precision, and therefore a high quality, for both tasks. If AL_1 represents the atomic allocation where the tasks were allocated to g at the same time (i.e. at_{sal} and at_{oxy} are performed concurrently by g), g must split its power between the two tasks, and the precision, and so the quality, of both measurements is reduced.

$$allocql(\{at_{oxy}\}, AL_1) + allocql(\{at_{sal}\}, AL_2) > allocql(\{at_{oxy}, at_{sal}\}, AL_3)$$

To simplify the development of the algorithms in this chapter we make the assumption that the quality of composite task completion is simply the sum of the qualities returned from its allocations of atomic tasks. This assumption is reasonable if;

1. there is a minimal effect on quality due to variations in a parent agent's aggregation of atomic tasks for different composite tasks;
2. the source of a parent agent's composite tasks does not affect the quality of its completion;
3. the quality of atomic tasks within a composite task are not dependent on each other.

We remove Assumption 3 once we have done the additional theoretical work in Chapter 10 to allow for the interdependence of atomic task qualities within a composite task.

System utility Over a period of time the system will progress through a number of system states as it allocates and completes tasks, from which we can define the utility of the system.

Definition 9.3.3 (*System utility*). The *utility* of a system is the sum of atomic allocation qualities of each allocation in a set of system states, $utility\colon 2^{\mathcal{S}} \to \mathbb{R}$ so that:

$$utility(S) = \sum_{\langle G_S(g), CL, AL, ACT, \phi \rangle \in S} allocql(atomics(AL), AL) \qquad (9.3)$$

In defining the utility, we assume that; the atomic tasks allocated in the set of states S are completed within that set of states; atomic tasks are instantaneously completed on allocation (and removed from the allocation). These are assumptions to simplify the definition of utility, and do not limit its applicability to our work overall.

Optimality of allocations

The range of allocations that an agent can achieve is bounded by its neighbourhood. The allocation of tasks AT' by agent g, to a neighbourhood of agents G' may be;

- *non-optimal*, there are other allocations of AT' to G' that will result in a higher allocation quality. Given an allocation AL' of atomic tasks AT' to a set of agents G' by an agent g:

$$\exists AL'', \forall \langle at, g'', g\rangle \in AL''\ \exists\, g'' \in G' : \mathit{allocql}(AT, AL'') > \mathit{allocql}(AT, AL')$$

- *locally-optimal*, the allocation achieves the highest quality possible given the atomic tasks AT' and agents G'. Given an allocation AL' of atomic tasks AT' to a set of agents G' by an agent g:

$$\nexists AL'', \forall \langle at, g'', g\rangle \in AL''\ \exists\, g'' \in G' : \mathit{allocql}(AT, AL'') > \mathit{allocql}(AT, AL')$$

- *system-optimal*, there is no other neighbourhood in the system that g could have that would produce a higher quality given the tasks AT'. Given an allocation AL' of atomic tasks AT' to a neighbourhood G' by an agent g, and any other possible neighbourhood of g, G'':

$$\nexists AL'', \forall \langle at, g'', g\rangle \in AL''\ \exists\, g'' \in G'' : \mathit{allocql}(AT, AL'') > \mathit{allocql}(AT, AL')$$

- *non-allocable*, the agents in the existing neighbourhood do not have the necessary capabilities to complete one or more of the tasks in AT':

$$\exists at \in AT' : \mathit{type}_a(at) \notin c(g) \forall g' \in N(g)$$

Note, there may be more than one allocation (and therefore neighbourhood) that is locally-optimal, or system-optimal. In the following equations where *argmax* is used to find optimal allocations, and returns multiple, degenerate values, we arbitrarily select a single member of the returned set. I.e. if $\underset{x \in X}{\mathit{argmax}}\ f(x)$ returns a set S we then select a member $x \in_R S$.

Definition 9.3.4 (*Locally-optimal allocation*)**.** The *locally-optimal allocation* of tasks AT to agents G given a fixed allocation of other tasks AL, is the allocation that gives the highest quality, $\mathit{optalloc}\colon 2^{\mathcal{AT}} \times 2^{\mathcal{G}} \times \mathcal{AL} \rightarrow \mathcal{AL}$, where:

$$\mathit{optalloc}(AT, G, AL) = \underset{AL' \in \mathit{permutations}(AT, G)}{\mathit{argmax}} \mathit{allocql}(\mathit{atomics}(AL'), AL \cup AL') \quad (9.4)$$

The quality of this locally-optimal allocation is then $\mathit{optallocql}\colon 2^{\mathcal{AT}} \times 2^{\mathcal{G}} \times \mathcal{AL} \rightarrow \mathbb{R}$:

$$\mathit{optallocql}(AT, G, AL) = \mathit{allocql}(AT, AL') \text{ where } AL' = \mathit{optalloc}(AT, G, AL) \quad (9.5)$$

Given the agents in the system, we can define all possible neighbourhoods for a given agent as $\mathit{allhoods}\colon \mathcal{G} \times 2^{\mathcal{G}} \rightarrow 2^{2^{\mathcal{G}}}$, where:

$$\mathit{allhoods}(g, G) = \{GS \in 2^G : |GS| < \delta_N\} \text{ given } g = \langle ..., \delta_N, ...\rangle$$

Of all these possible neighbourhoods, there will be one or more that will give the locally-optimal allocation with the highest quality within the system for an agent's allocation of a set of tasks.

Definition 9.3.5 (*Optimal neighbourhood*)**.** In a system containing agents G with allocation AL, the *optimal neighbourhood* of a set of atomic tasks AT, allocated to an agent g, will be that which gives the best quality $opthood\colon 2^{\mathcal{AT}} \times \mathcal{G} \times 2^{\mathcal{G}} \times \mathcal{AL} \rightarrow 2^{\mathcal{G}}$ where:

$$opthood(AT, g, G, AL) = \underset{G' \in allhoods(g,G)}{argmax} \quad optallocql(AT, G', AL) \tag{9.6}$$

The *system-optimal allocation*, $sysalloc\colon 2^{\mathcal{AT}} \times \mathcal{G} \times 2^{\mathcal{G}} \times \mathcal{AL} \rightarrow \mathcal{AL}$ is therefore the locally-optimal allocation to this neighbourhood:

$$sysalloc(AT, g, G, AL) = optalloc(AT, opthood(AT, g, G, AL), AL) \tag{9.7}$$

Where the quality of this allocation will be, $sysallocql\colon 2^{\mathcal{AT}} \times \mathcal{G} \times 2^{\mathcal{G}} \times \mathcal{AL} \rightarrow \mathbb{R}$, such that:

$$sysallocql(AT, g, G, AL) = allocql(AL') \text{ where } AL' = sysalloc(AT, g, G, AL) \tag{9.8}$$

If we had full knowledge of the system's tasks AT and agents G, we could then calculate the *joint-optimal allocation*, the global allocation with the highest quality, the *joint-optimal quality*, given by $jointallocql\colon 2^{\mathcal{AT}} \times 2^{\mathcal{G}} \rightarrow \mathbb{R}$:

$$jointallocql(AT, G) = optallocql(AT, G, \emptyset)$$

Our aim is to optimise towards this value using distributed algorithms, where only an agent's local-knowledge is used.

Example 9.3.2 (*Optimal allocations in multi-agent systems*)**.** In Figure 9.1 we illustrate locally-optimal, system-optimal, and joint-optimal allocations in a simple multi-agent system. A parent agent, pg_1, allocates two tasks, $\{at_1, at_2\}$, with type $\{ap_1, ap_2\}$ in a system with 3 child agents cg_1, cg_2 and cg_3. Each agent can complete tasks of these types to different task qualities, $atomicql(ap, |concurrent(AL, g)|)$.

If the neighbourhood of pg_1 is N_1, then, due to the effect of concurrency the task completion qualities of child agents, the locally-optimal allocation for pg_1 in that neighbourhood is $\{\langle at_1, cg_1\rangle, \langle at_2, cg_2\rangle\}$, with value 6. If pg_1 changed to neighbourhood N_2, then the quality of its task allocations to child agents cg_2 and cg_3 is the best possible in the system (if it were the only agent allocating tasks). So N_2 is its optimal neighbourhood for pg_1 given these tasks to allocate, and its system-optimal allocation would be $\{\langle at_1, cg_3\rangle, \langle at_2, cg_2\rangle\}$, with quality 9. However, in a multi-agent system, parent agents may independently choose to allocate tasks to shared child agents. If pg_2 were also to allocate a task at_1' of type ap_1 to cg_3, concurrency would cause the quality returned to pg_1 for at_1 to drop to 1, so the actual joint-optimal allocation[1] in this situation is $\{\langle at_1, cg_1\rangle, \langle at_2, cg_2\rangle, \langle at_1', cg_3\rangle\}$, giving a quality of 11.

[1] Note that multiple degenerate joint-optimal allocations may be possible in a system.

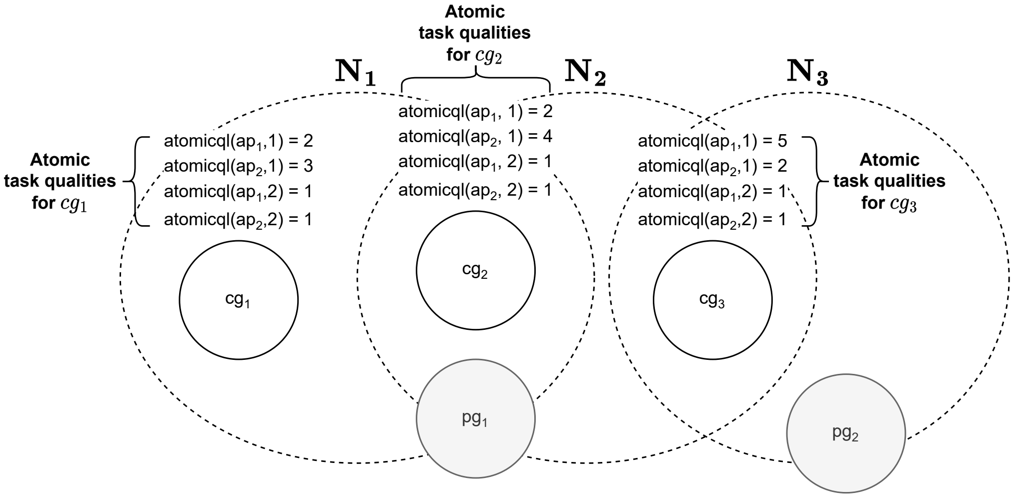


Figure 9.1: Types of optimal task allocations in a multi-agent system. *The diagram illustrates how tasks can be allocated within a system to attain local, system, and joint-optimal solutions. We show each agent's possible atomic task qualities for different atomic task types and concurrent allocations.*

Finding the optimal task allocation

Given a set of agents and a set of composite tasks, how can we then optimise towards the joint-optimal allocation[2], maximising the utility of the system? How do we do this when the capabilities of the agents, and the quality to which each agent can complete atomic tasks, are dynamic and unknown? We can view this as two main sub-problems[3];

1. given a fixed local neighbourhood, how can an agent find the locally-optimal allocation of atomic tasks from a sequence of composite tasks that are assigned to it over a period of time?
2. how does an agent find the optimal neighbourhood within the set of all possible neighbourhoods it can achieve, containing the system-optimal allocation for a set of atomic tasks?

9.4 Adapting Q-learning techniques for non-stationary environments

For all possible actions an agent can take there is a likelihood that taking that action in the current state will increase future rewards. When an action is taken, the accuracy of these predicted values can be improved based on the actual rewards returned. This is how an agent can improve its *policy*, the mapping of its view of the state of the system, and the actions it can take within those states, to the likelihood those actions are

[2] We use the joint-optimal allocation as the maximal system performance to compare our algorithms against in Section 9.6.

[3] This could be viewed as a bi-level optimisation problem[296] where multiple objectives are involved, one objective (the optimisation of an agent's task allocations) being nested inside another (the optimisation of an agent's neighbourhood within the system). Relevant work that could be adaptable to this use case can be found for intrinsic reward optimisation[297], and actor-critic reinforcement learning[298].

optimal. Q-learning methods have been successfully applied as a model-free method of learning policies[71], however, in real-world multi-agent systems, we commonly find state-spaces that are only partially-observable to agents, and are non-stationary, where Q-learning is more complex to apply[73], [299].

Policy drift in non-stationary environments

In stationary environments an agent's policy targets a single Markov Decision Process (MDP) that does not change[300], [301]. Over time, actions an agent believes to be non-optimal become increasingly unlikely to be updated. With multiple agents interacting however, the environment is now non-stationary, and becomes an infinite, and difficult to predict, sequence of MDPs[302]. Without sampling actions previously judged to be non-optimal, an agent's policy will not adapt. Unless it can detect changes in the sequence of MDPs, and explore new actions, it becomes stuck applying a previous policy to increasingly different MDPs[303].

To adapt reinforcement learning algorithms for these systems, we can use experience replay, updating the learning algorithms with the rewards for previously taken actions again, with the repetition frequency being inversely proportional to their likelihood of being chosen[304], or reset values periodically to restart the learning process[305]. However, these approaches do not take account of changes in the rate of drift of the optimal policy during the lifetime of the system, or if the change is transient (e.g. temporary weather conditions). Our work tackles this problem in the following ways[4];

1. an agent will optimise for its partially observable state using temporal difference-based reinforcement learning by default (as described next in this section);
2. if current rewards are good compared to historical values, the ATA-RIA algorithm we develop assumes a relatively stationary state, and continues reinforcement learning updates to optimise the agent's policy;
3. if rewards are historically poor, the RT-ARP algorithm will increase the likelihood of choosing non-optimal actions, updating them more frequently. This likelihood (and how it varies between action-categories depending on the scale of their impact on an agent's policy) changes with the scale of the difference with those historical rewards (See Sections 9.4.3 - 9.4.5);
4. as the policy changes, previous knowledge becomes less valued, and is forgotten, by the SAS-KR algorithm, gradually resetting learning.

9.4.1 Q-learning

When an agent g carries out an action a in system state s, then there is an expected value, or *cumulative discounted reward* to doing so[5], given by the optimal Q-mapping $Q(g)^* : \mathcal{S} \times 2^{\mathcal{A}} \rightarrow 2^{\mathbb{R}}$. With no initial knowledge of the environment, an agent instead can learn estimated values for each action a in a state S by maintaining a *Q-table*, a table of estimates, $Q(g) : \mathcal{S} \times 2^{\mathcal{A}} \rightarrow 2^{\mathbb{R}}$, where $Q[s, A]$ selects the estimated values for set of actions A, in state s[6]. The agent then takes actions, and updates

[4] See Section 9.5 for more details on the ATA-RIA, RT-ARP, and SAS-KR algorithms.

[5] The cumulative reward is the expected rewards the agent could receive if it continued to take the optimal actions in the future starting from its next state $s^{'}$. The discount factor γ makes the agent optimise for shorter time horizons over longer ones, with the assumption that there is more uncertainty about the system state in the future than the near-term.

[6] Note, individual estimated values for a single action can be returned by $Q(g)[s, a]$.

these estimate values through temporal difference learning (as described in Section 5.4). These Q-table values allow the agent to choose its estimated best action given its current state.

Updating a Q-table

In taking an action, an agent g will receive a reward, r, and the system will move from state s to state $s^{'}$, where the optimal action for g to take would be $a^{'}$. On the change of state, the agent updates its Q-table estimate for state s and action a using the formula below[7]:

$$Q(g)^{'}[s,a] \leftarrow \underbrace{Q(g)[s,a]}_{\text{old value}} + \underbrace{\alpha}_{\substack{\text{learning}\\\text{rate}}} \overbrace{\Bigg(\underbrace{\underbrace{r}_{\text{reward}} + \underbrace{\gamma}_{\substack{\text{discount}\\\text{factor}}} \underbrace{\max_{a'} Q(g)[s^{'},a^{'}]}_{\substack{\text{estimate of}\\\text{optimal}\\\text{future value}}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(g)[s,a]}_{\text{old value}} \Bigg)}^{\text{temporal difference}} \tag{9.9}$$

Learning rate α and discount factor γ constants

The *learning rate* α and *discount factor* γ are constants set at system initialisation. The *learning rate* $0 < \alpha \leq 1$ is a constant that controls the step-size of the update, corresponding to how quickly an agent will alter its policy based on the rewards it is receiving from its current actions. The *discount factor* $0 < \gamma \leq 1$ constant alters the value of rewards depending on how recent they are. Where a smaller value focuses the agent on learning to value actions with more immediate results, while with a larger value it will take into account the action's effects longer into the future.

This update process ignores the possibility that the optimal action policy can change, however as discussed, this is mitigated by the other steps introduced in Section 9.2 and detailed in the following subsections.

Reward values for actions

The reward for carrying out an action is dependent on the action taken. For EXEC and SINGLEALLOC actions this is simply the quality of task completion by the agent the task is allocated to, i.e, $r = allocql(\{at\}, AL)$. For INFO and LINK actions we set a fixed, negative reward value, representing the opportunity cost of an agent choosing these actions over actions to complete atomic tasks. We define the function $reward \colon A \rightarrow \mathbb{R}$ as the reward for an action a.

The RLUpdate function

To help simplify our algorithm definitions, we define the update function, $rlupdate \colon \mathcal{G} \times \mathcal{S} \times \mathcal{S} \times \mathcal{A} \times \mathbb{R} \times \mathcal{Q} \rightarrow \mathcal{Q}$, which takes an agent g's current state s, new state $s^{'}$, an action a, a reward value r, and the agent's Q-table Q, and returns the Q-table as updated by Equation 9.9.

9.4.2 The value of knowledge

Action samples

In order to use an agent's historical performance to alter its future behaviour we need to explicitly store information on past actions and their outcomes. We do this by defining an agent's *action samples*, a set of tuples $sp = \langle a, t, r \rangle$ for each agent, where a is an action taken at time t that gave reward r. We define the *action sample selection*

[7] This standard Q-learning algorithm is adapted from the version described in Wikipedia[306], as introduced by Watkins[168].

function *samples*: $2^{\mathcal{SP}} \times 2^{\mathcal{A}} \longrightarrow 2^{\mathcal{SP}}$ to allow us to specify subsets of action samples SP for which the action performed was an element of a given set of actions, A:

$$samples(SP, A) = \{\langle a, t, r\rangle : \ \forall \langle a, t, r\rangle \in SP, a \in A\} \tag{9.10}$$

For convenience, we also define the *latest sample time* in a set of samples, $latest: \mathcal{SP} \longrightarrow \mathbb{R}$, as a function that takes a set of actions samples returns the time of the most recent sample.

Information value We first make an assumption that the more recently, and frequently, an agent takes an action, the less uncertainty it has in predicting that action's contribution to longer-term rewards. If the knowledge learned about an action's contribution is highly uncertain, then we judge its loss to have minimal negative effect to the agent's performance, and to be quickly re-learnable by the agent to the same or greater level of value.

The *action information quality*, $actval: 2^{\mathcal{SP}} \times 2^{\mathcal{A}} \times \mathbb{N}_0 \longrightarrow \mathbb{R}$, is a proxy for the value of information collected about an action a up to a time t, given the set of action samples SP:

$$actval(SP, A, t) = \frac{1}{|A|} \sum_{a \in A} \frac{|samples(SP, \{a\})|}{t - latest(samples(SP, \{a\}))} \tag{9.11}$$

A constant can then be chosen, the *uncertain information threshold* $\hat{\mu}_{\mathtt{min}}$, as the minimum value below which an agent's information about the expected rewards of taking an action is no longer considered useful for prediction.

The value of neighbourhood agents We define *neighbour information value* $hoodval: 2^{\mathcal{SP}} \times \mathcal{G} \times 2^{\mathcal{G}} \longrightarrow \mathbb{R}$ as the sum of the quality values of all action samples SP of an agent g that refer to actions that involve agents in a set G:

$$hoodval(SP, g, G) = \sum_{\langle a,t,r\rangle \in SP,\ a \in targets(g,s,G)} r \tag{9.12}$$

Definition 9.4.1 (*Minimum value neighbour*)**.** The *minimum value neighbour* of an agent g is the child agent that generates the least neighbour information value, $minhoodval: 2^{\mathcal{SP}} \times \mathcal{G} \longrightarrow \mathcal{G}$:

$$minhoodval(SP, g) = \underset{x \in N(g)}{argmin}\ hoodval(SP, g, \{x\}) \tag{9.13}$$

9.4.3 Predicting the effect of actions

Taking actions of different categories will change an agent's neighbourhood, N, or knowledge base, K to differing degrees. Predicting how, and to what extent, each action will impact an agent's current policy is key to adapting Q-learning to handle the non-stationary environment as the agent can then combine its prediction of how much and how quickly it should change its policy (to be covered in Sections 9.4.4 and 9.4.5), with the selection of actions that will enable it to do so.

To enable agents to make these predictions we;

1. define the impact of the different categories of actions on both an agent's neighbourhood and knowledge;
2. estimate the probability that actions generating impact will actually occur;

3. combine these factors to define action impact;
4. detail algorithms based on historical quality values to predict which action impacts will have a positive effect on task completion quality.

Neighbourhood and knowledge impacts

There is an impact on possible allocation quality if an agent takes actions that change its neighbourhood. This *neighbourhood impact* on an agent allocating atomic tasks AT from changing its neighbourhood from $N^{'}$ to $N^{''}$ within a system with allocation AL is the difference between the locally-optimal allocation qualities of the respective neighbourhoods, *hoodimpact*: $2^{\mathcal{AT}} \times 2^{\mathcal{G}} \times 2^{\mathcal{G}} \times \mathcal{AL} \rightarrow \mathbb{R}$, where:

$$hoodimpact(AT, N^{'}, N^{''}, AL) = optallocql(AT, N^{''}, AL) - optallocql(AT, N^{'}, AL) \tag{9.14}$$

Definition 9.4.2 (*Maximum neighbourhood impact*)**.** The *maximum neighbourhood impact maxhoodimpact*: $2^{\mathcal{AT}} \times 2^{\mathcal{G}} \times \mathcal{AL} \rightarrow \mathbb{R}$ is the maximum possible neighbourhood impact given a set of atomic tasks AT and all combinations of neighbourhoods that can be formed from a set of agents G:

$$maxhoodimpact(AT, G, AL) = \underset{\forall \langle X,Y \rangle \subseteq \langle 2^{G} \times 2^{G} \rangle}{argmax} \quad hoodimpact(AT, X, Y, AL) \tag{9.15}$$

Definition 9.4.3 (*Knowledge impact*)**.** The *knowledge impact* of an agent changing its knowledge from set of knowledge $K^{'}$ to $K^{''}$ is the difference between the maximal neighbourhood impacts, *knowimpact*: $2^{\mathcal{AT}} \times 2^{\mathcal{G}} \times 2^{\mathcal{G}} \times \mathcal{AL} \rightarrow \mathbb{R}$, where:

$$\begin{aligned} knowimpact(AT, K^{'}, K^{''}, AL) = {} & maxhoodimpact(AT, K^{''}, AL) \\ & - maxhoodimpact(AT, K^{'}, AL) \end{aligned} \tag{9.16}$$

Example 9.4.1 (*Neighbourhood impact in a WSN*)**.** An agent g in an ocean monitoring WSN system has a knowledge base from which it can form 3 distinct neighbourhoods; N_1, which is the current neighbourhood, N_2, and N_3. g needs to allocate an oxygen reading task, at_{oxy}. The locally-optimal allocation quality of N_2 is worse than that of N_1 (e.g., due to low battery levels of agents in N_2), whereas that of N_3 is much better. In this case, if g was to take an action to replace N_1 with N_2, then this would give $ni(\{at_{oxy}\}, N_1, N_2, AL) < 0$, a negative impact. In contrast, taking an action that replaces N_1 with N_3 would give $ni(\{at_{oxy}\}, N_1, N_3, AL) > 0$, which is then the maximum neighbourhood impact, given the knowledge base $N_1 \cup N_2 \cup N_3$.

The probability of impact effects

Since neighbourhoods and knowledge are dynamic, agents are continually added and removed from both sets. Therefore there is a probability that agents in a neighbourhood never contribute to the quality of a composite task before they are removed or the task is completed. In other words, when an agent moves from a neighbourhood $N^{'}$ to $N^{''}$, it will lose access to actions involving agents in the set $N^{'} - N^{''}$, and gain access to actions involving those agents in the set $N^{''} - N^{'}$. If actions due to agents in those sets are never taken, there is no overall impact to allocation qualities on changing the neighbourhood or knowledge base. We define the probability an agent takes actions involving its neighbourhood agents in the sets $N^{'} - N^{''}$ or $N^{''} - N^{'}$ as the *neighbourhood impact probability*, $P(N^{'}, N^{''})$. Similarly we define the *knowledge impact probability* as $P(K^{'}, K^{''})$.

Estimating the impact of taking an action

The *action impact* is the expected value of the change in possible allocation quality if an action a is taken. On taking the action the neighbourhood is changed from $N^{'} \rightarrow N^{''}$ and the knowledge base from $K^{'} \rightarrow K^{''}$. We define this function $actimpact\colon 2^{\mathcal{AT}} \times 2^{\mathcal{G}} \times 2^{\mathcal{G}} \times 2^{\mathcal{G}} \times 2^{\mathcal{G}} \times \mathcal{AL} \rightarrow \mathbb{R}$, where:

$$\begin{aligned} actimpact(AT, N^{'}, N^{''}, K^{'}, K^{''}, AL) = P(N^{'}, N^{''}).hoodimpact(AT, N^{'}, N^{''}, AL) \\ + P(K^{'}, K^{''}).knowimpact(AT, K^{'}, K^{''}, AL) \end{aligned} \tag{9.17}$$

The probability an agent will take different actions, and how those types of actions will impact the possible qualities of the atomic task completions in a changed neighbourhood or knowledge base, are generally unknown and difficult to predict in complex, dynamic systems. In order to calculate action-impact values we therefore make some simplifications, allowing us to estimate these values. Using estimations allows us to focus on the core algorithm behaviours without adding complexity.

The estimations are chosen such that the different types of action are separable based on their impact values. Since these values are combined with dynamically learned values in the RT-ARP algorithm, as long as the estimated action-impact values are ordered the same way as the actual values, the algorithms should work as expected. The more accurate the estimations, the more quickly we should expect learning to proceed, and so implementing more granular impact prediction in future work could be expected to improve the algorithms' performance.

As a first approximation, we estimate action-impacts $\widehat{ai}$, based on whether each action-category changes the state of neighbourhoods or knowledge bases, and the probability of the impacts described given a system's size. We detail how these estimations are calculated in Appendix C.3.

The set of *action-impact values*, W, are estimated values for maximum action impacts for each action-category, $category(a)$. We assume that both $|N^{''} - N^{'}| \in \{0, 1\}$ and $|K^{''} - K^{'}| \in \{0, 1\}$ for all actions. We also assume that the size of the overall system allocation of atomic tasks, AL, is large enough to remain approximately constant despite any allocation change or resource pressure resulting from the action.

$$W = \{\langle category(a), \widehat{actimpact}(AT, N^{'}, N^{''}, K^{'}, K^{''}, AL)\rangle : \forall a \in A\} \tag{9.18}$$

9.4.4 Measuring relative allocation optimality

For an agent to know the optimal task quality it could achieve in its current neighbourhood we use a metric to measure how far its current quality values are from optimal.

Definition 9.4.4 (*Locally-optimal allocation metric*)**.** The *locally-optimal allocation metric localdist*$\colon 2^{\mathcal{AT}} \times \mathcal{G} \times \mathcal{AL} \rightarrow \mathbb{R}$ is the difference between an agent's current allocation quality of atomic tasks AT to agents in its neighbourhood, and the locally-optimal allocation quality:

$$localdist(AT, g, AL) = optallocql(AT, N(g), AL) - allocql(AT, AL) \tag{9.19}$$

Definition 9.4.5 (*System-optimal allocation metric*)**.** The *system-optimal allocation metric systemdist*$\colon 2^{\mathcal{AT}} \times \mathcal{G} \times 2^{\mathcal{G}} \times \mathcal{AL} \rightarrow \mathbb{R}$ is the difference between an agent's current allocation quality and the system-optimal allocation quality given the set of agents in the system, G:

$$systemdist(AT, g, G, AL) = sysallocql(AT, g, G, AL) - allocql(AT, AL) \tag{9.20}$$

9.4.5 Predicting impact from historical performance

An agent g allocating a set of atomic tasks AT in a system of agents G with allocation AL, wants to know before taking an action a that will change its neighbourhood N or knowledge K, if doing is likely to improve its future performance in allocating AT. To predict this likelihood, we define a function $positiveact\colon \mathcal{A} \times 2^{\mathcal{AT}} \times \mathcal{G} \times 2^{\mathcal{G}} \times \mathcal{AL} \to \mathbb{R}$ that maps these variables to the probability that the outcome of action a will be positive in terms of the future atomic allocation quality of g's allocation of atomic tasks AT.

Predicting beneficial actions

Knowing the value of *positiveact* would depend on an agent having knowledge of *localdist* and *systemdist*, however, with only partial-knowledge of the non-stationary environment, g does not know the locally-optimal allocation of AT in its current neighbourhood, or the system-optimal allocation of AT overall. Therefore g cannot directly calculate if taking an action a that changes its N or K would be beneficial.

In this subsection we use action-samples (Subsection 9.4.2), over multiple time-scales, as a proxy for the values of *localdist* and *systemdist* (Subsection 9.4.4). Combining these proxy values with the impact of actions (Subsection 9.4.3) allows us to formulate the *impact transformation function* as a method of estimating *positiveact*.

Assumptions in using historical rewards

An agent needs to know the locally-optimal allocation quality for both the current and the future neighbourhoods to predict whether the impact of changing neighbourhoods from $N^{'}$ to $N^{''}$ would be positive. This is difficult since the agent is uncertain of *localdist* and so does not know the best values it can obtain in the current neighbourhood. However, it is likely to have less samples of the actions available in $N^{''}$ so may have even more uncertainty in future values if it changed neighbourhoods. To find proxies for these values we make the following assumptions based around time-based trends in action-samples.

Assumption 1. (*Likelihood of neighbourhood change*) The more actions an agent takes the greater the likelihood that it will have taken actions that change its neighbourhood.

If there is always some exploration of the action-space this assumption is reasonable. In our algorithms we utilise Boltzmann selection with a fixed temperature so this holds true. In some annealing-based learning algorithms, exploration of the action space will decrease over time and this may not hold true. However, in dynamic systems these non-adaptive, time-based approaches would not be applicable in any case.

Assumption 2. (*Variation of neighbourhoods*) Samples in a large set of historical action-samples will come from many different neighbourhoods.

Making this assumption allows an agent to compare its current performance with historical values and assume it represents a statistical comparison of its current neighbourhood to others in the system. Where an agent has access to all its possible neighbourhoods, our algorithms should find the system-optimal neighbourhood for that agent. If the agent can only access a small subset of neighbourhoods, it should find the best in that subset. As such, the algorithms should perform well in both scenarios.

Assumption 3. (*Time-dependent similarity of neighbourhoods*) Action-samples separated by short spaces of time are likely to be from similar neighbourhoods. Those separated by large amounts of time are more likely to represent very different neighbourhoods.

Using this assumption, if an agent has had much better rewards in the past, it can use these to infer that its current neighbourhood might benefit from being substantially changed to improve performance. However, in systems where agents can only reach a very limited possible set of neighbourhoods (e.g. due to their static location and limited broadcast range), this may not be reasonable. In such systems our algorithms would push agents to take risky actions that substantially alter their neighbourhoods and knowledge, when exploitation of the current one may be the better choice.

By making these assumptions we can estimate the relative locally-optimal allocation and system-optimal allocation metric values. As recent action-samples with small-time separations come from the same or similar neighbourhoods we compare their quality value statistics to estimate $localdist$. As action-samples over the long-term come from many different neighbourhoods, we compare their values to estimate $systemdist$.

Methods to estimate action-impacts

To predict which actions will have a positive impact we firstly use historical action-sample quality values to estimate action-impacts. Based on these values we increase or decrease the probabilities of taking different categories of action. Whether an impact is estimated to be positive or negative will alter the agent's likelihood of taking actions that explore allocation within the current neighbourhood or change its neighbourhood or knowledge base[8]. The process is as follows;

1. we define the *time-summarised quality matrix (TSQM)*, a method of summarising historical quality returns over multiple time scales. This uses a resampling technique where each row in the matrix is the result of downsampling the time-series data of the previous row[307]. The update period for each row's recalculation, and the frequency of downsampling is dictated by the matrix dimensions (see Section 9.4.5);
2. using this matrix we generate the *impact interpolation function*;
3. we then define the *impact transformation function* using a ratio of integrations over the impact interpolation function;
4. finally we use the action-impact values for each action-category that that will be used to as the input for the impact transformation function.

[8] Note, this has similarities to policy-based reinforcement learning strategies. In this case however, the policy targets broad categories of actions based on how they are predicted to affect the agent's knowledge. The policy is informed by the agent's prediction of its possible performance within its neighbourhood, as well as the predicted quality of that neighbourhood overall within the system, which is then used to alter the actions chosen through the agent's value-based strategy.

Time-summarised quality matrix (TSQM)

A TSQM Λ has shape $(m \times n)$ with all values initially null[9]. Time-ordered actions-sample reward values $\{r_t, r_{t-1} \ldots, r_{t-n}\}$ for all actions of a specific agent are added to the first row $\Lambda_{(0,j)}$ as they are sampled such that, $\Lambda_{(0,)} \longleftarrow \{r_i\}_{i=0}^{n}$. Each subsequent row is the result of averaging and pooling values in the previous row. This approach allows each row to represent the quality trends across different time-scales. If h is the number of quality values added to the matrix then we update the elements as follows:

$$\Lambda_{(i+1,k)} \longleftarrow \frac{\sum \Lambda_{(i,)}}{\left|\Lambda_{(i,)}\right|}, \text{ if } h \bmod \left(k\left|\Lambda_{(i,)}\right|\right) = 0 \tag{9.21}$$

To summarise the process of updating the matrix,

- each new value of r updates the first cell of the initial row of Λ, row 0. As this cell is updated, the other values in the row are moved along by one to accommodate it, with the last value being discarded;
- after n new values of r have been added to row 0, an average of row 0 is taken and added as the first cell of the row 1. All values on row 1 are moved along by one, and the last value discarded. The same process will happen after the addition of each batch of n, r values to row 0;
- after n new average values have been added to row 1, row 2 will have the average of row 1 added to its first cell, moving all the others along and discarding the last;
- the same process continues for all other rows. Where each will update its first cell with the average of the previous row after n values have been added to that row.

We use the function $\texttt{updatetsqm}(\Lambda(g), r)$ as shorthand for the full update process of the TSQM of an agent g, denoted by $\Lambda(g)$. Note that the values of $(m \times n)$ will alter the behaviour of the TSQM in the following ways. Increasing the value of m will increase the number of trends over different timescales that the agent will use. Whereas the larger the value of n, the greater the number of quality values the agent must receive before it updates each of these longer-term trends.

Impact interpolation function

The *impact interpolation function*, $\mathit{impactinterpol}(x)$, is generated by taking a linear interpolation[10] over the rows of a TSQM (see Figure 9.2). A decay factor $\delta \in [0, 1]$ is chosen to dampen the values of longer time-scales (exponentially by the row exponent i) to allow more recent trends to have a stronger impact. For a TSQM of shape $(m \times n)$ a value $x \in \mathbb{R}[0, 1]$ will be transformed as:

$$\mathit{impactinterpol}(x) = \text{interpol}\left[\left\{\left(\frac{i}{N}, \text{average}(\Lambda_{(i,)})\delta^i\right)\right\}_{i=0}^{N}\right](x), \text{ for layers } 0 \text{ to } N \tag{9.22}$$

The effect of this is to dynamically generate a function from the TSQM matrix where a parameter x will be mapped to reward-value trends, with larger values of x mapping to longer-term trends.

[9] Where m and n are values chosen at system initialisation.

[10] We use a 1-d linear interpolation, implemented using the python `scipy.interpolate.interp1d` method. Where $\text{interpol}[\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}](x) = y$ estimates the value y, from x, using an interpolated function generated from a known set of values, $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$.

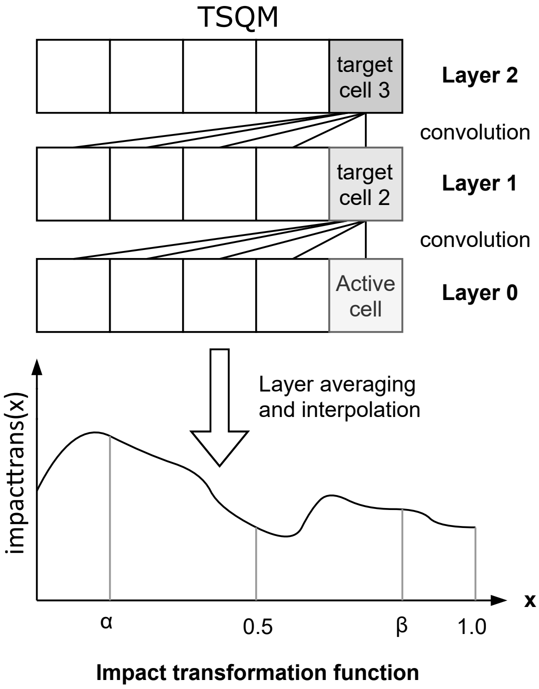


Figure 9.2: Transforming the TSQM. *The impact transformation function is generated from the TSQM through a process of convolution of prior rows in the matrix, followed by interpolation over the rows as a whole.*

Impact transformation function The *impact transformation function* estimates the probability that taking an action from an action-category in the current neighbourhood will be positive by taking a ratio over the integrals of the interpolation representing the fraction of historical quality values that occur up to the input value. For any $y \in \mathbb{R}[0,1]$ this is given by:

$$impacttrans(\mathtt{w}) = 1 - \frac{\int_{y=0}^{\mathtt{w}} impactinterpol(y)\, dy}{\int_{y=0}^{1} impactinterpol(y)\, dy} \tag{9.23}$$

Given a value x, if $it(\mathtt{w})$ is close to 0, then the quality values generated by the system have been better than over the time period represented by the range $[\mathtt{w},1]$, than over the shorter time period $[0,\mathtt{w}]$. This shows the system's near-term performance is worse than its longer-term performance. Conversely, if $it(\mathtt{w})$ is close to 1, then the short-term performance of the system is better than its previous longer-term trends.

Impact exploration factor We use this balance of the impact transformation function between shorter and longer timescales to adapt the exploration behaviour of our reinforcement learning model. We choose a constant k_{ief} at system initialisation[11] to define the *impact exploration factor*: $\epsilon_{ief} = impacttrans(k_{ief})$. Higher values of ϵ_{ief} mean the agent is attaining better performance now than in the past and should exploit rather than explore the system further. Lower values mean its exploration of the system should be increased[12].

[11] We use the midway value of $k_{ief} = 0.5$, as an impact exploration factor for our simulations.

[12] We multiply this by a constant factor ϵ_{base} in the RT-ARP algorithm to ensure a useful scale on exploration (See Algorithm 2).

Finally we can use the interpolation of action-impact values $\mathtt{w}$ of action-categories of each action a to estimate the probability that taking those type of action will have a positive impact,

$$P(hoodimpact(AT, N^{'}, N^{''}, AL) > 0 \mid a) \approx impacttrans(\mathtt{w}) \tag{9.24}$$

Using this method, agents will prefer lower-risk actions when the system is performing well, $impacttrans(\mathtt{w}) \rightarrow 0$, and higher-risk actions when the systems performing historically poorly, $impacttrans(\mathtt{w}) \rightarrow 1$.

9.4.6 Extending the agent state for task allocation

In Chapter 8, Section 8.3 we defined the state of an agent in terms of its neighbourhood and knowledge only. We now extend this definition to incorporate the agent's action samples (Section 9.4.2), TSQM (Subsection 9.4.5), and Q-table (Subsection 9.4.1).

Definition 9.4.6 (*Agent State (Extended)*)**.** Given an agent $g = \langle id, c, r, \delta_k, \delta_n \rangle$, we define its state as a tuple $\langle K, N, SP, \Lambda, Q \rangle$, where:

- $K \subseteq G$ is the knowledge of the agent.
- $N \subset K$ is the neighbourhood of the agent.
- SP is the set of action samples of the agent.
- Λ is the TSQM of the agent.
- Q is the Q-table of the agent.

As with the notation for an agent's neighbourhood $N(g)$ and knowledge $K(g)$ used in our definition of agent state in Chapter 8, Section 8.3, we will use the notation $SP(g)$, $\Lambda(g)$, and $Q(g)$ to refer to the action samples, TSQM, and Q-table, respectively, of an agent g.

9.5 Algorithms for optimal task allocation

We now introduce our algorithms for solving the task-allocation problem. The solution presented in this section allows an agent to determine the capability of other known agents to perform tasks, to allocate these tasks, and to carry out other actions based on the agent's current knowledge and the need to explore the capabilities of agents in the system.

Strategy of the algorithms

To tackle the problems as defined in Section 9.3, as well as handle the resource constraints of agents, we utilise two high-level approaches:

1. an agent should adapt its selection of actions based on how it good it judges its current performance to be. This judgement should take into account;
   (a) the agent's performance in its present neighbourhood compared to its past performance;
   (b) the agent's prediction of whether the locally-optimal allocation of its atomic tasks in its current neighbourhood (which it may not have found yet) is better or worse than that of previously experienced neighbourhoods;
   (c) the agent's prediction of the likelihood of neighbourhoods existing in the

system with locally-optimal allocations of its atomic tasks better than any it has experienced so far.

2. an agent should keep its resource usage within its constraints by only retaining the neighbourhood agents and knowledge it considers the most useful to its performance. In doing so, an agent will be able to find neighbourhoods where it performs well more efficiently as it does not need to learn from performing task and action allocations to less useful agents.

Purpose of the algorithms To achieve (1), we introduce the RT-ARP algorithm, for (2), we see the N-Prune and SAS-KR algorithms. To orchestrate the process and maintain the information needed for it to function, we use the ATA-RIA algorithm. This forms the basis of our first contribution (see Chapter 1, Contribution 1). At a high-level, the purpose of each of the algorithms introduced are;

- the *agent task allocation with risk-impact awareness (ATA-RIA)* algorithm enables each agent to choose a subset of other agents in the system based on how much it predicts those agents will help complete the atomic tasks that comprise the composite tasks that are allocated to them. They can learn the best task allocation strategy for these agents, but can also change which agents comprise the group to improve performance. It integrates the RT-ARP, SAS-KR, and N-Prune algorithms, as well as updating an agent's Q-table to reflect the rewards it receives for taking actions, and the action-sample data with these rewards;
- the *reward trends for action-risks probabilities (RT-ARP)* algorithm gives agents the ability to transform their exploration strategies given the trends in how well the atomic tasks they have allocated in the past have been performed. Using this algorithm, agents can increase or decrease the likelihood of them taking actions that may cause larger changes to their task allocation strategy. The algorithm increases the probability of an agent taking neighbourhood and knowledge-altering actions and increasing exploration when the possible allocation quality achievable in its current neighbourhood is relatively poor compared to previous neighbourhoods;
- the *state-action space knowledge-retention (SAS-KR)* algorithm implements a knowledge retention scheme under dynamic neighbourhood changes, optimising an agent's knowledge while keeping it within its size constraints. It selectively retains the most useful information the agent has learned about state-action space, and of its past actions, removing knowledge that is considered to have been less useful to the agent in optimising task allocation in the past;
- the *neighbourhood update (N-Prune)* algorithm selectively removes agents from the group considered for task allocation by an agent while keeping the agent's neighbourhood within its size limitations. This selection is based on not only how much the agent predicts the other agents will contribute to its composite task, but also how much uncertainty it has about that prediction.

Figure 9.3 shows the steps of the ATA-RIA algorithm at a high-level, and how the other algorithms are integrated into its workflow.

Standard functions In developing our algorithms we use some standard functions for normalisation, and selection, as shown in Tables 9.1, and 9.2 respectively. In addition we introduce the *agent wait* function, $wait(g)$, to set an agent g to wait for the state of the system to progress by one step of the system counter, i.e. $\phi \rightarrow \phi + 1$.

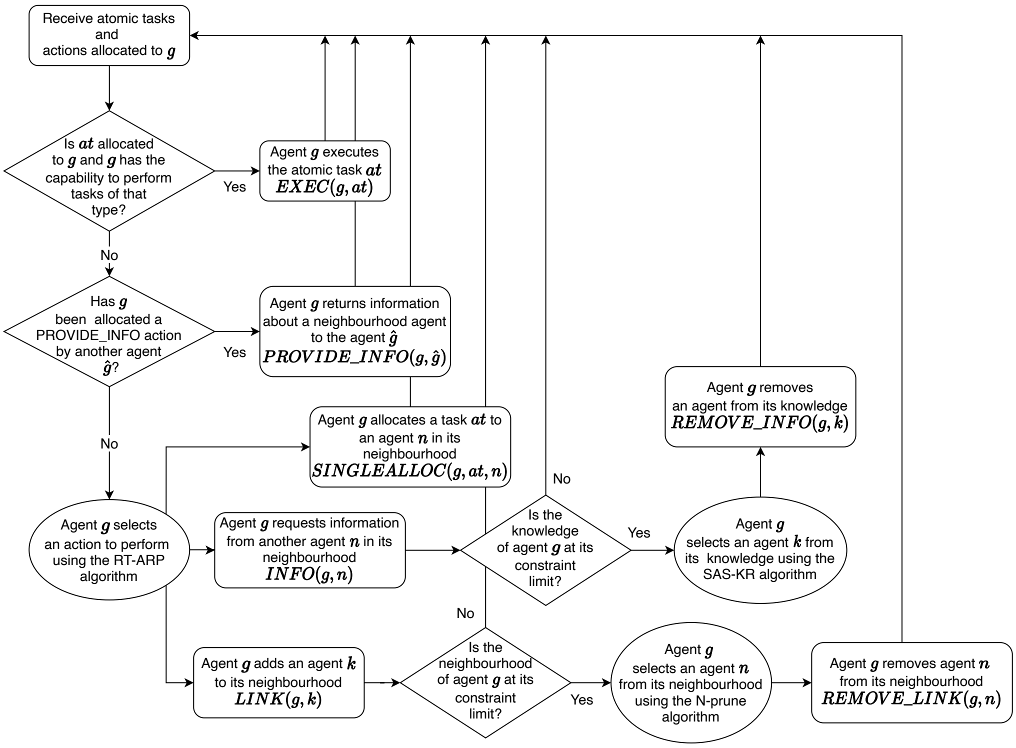

Receive atomic tasks and actions allocated to g
Is at allocated to g and g has the capability to perform tasks of that type?
Yes
No
Agent g executes the atomic task at $EXEC(g, at)$
Has g been allocated a PROVIDE_INFO action by another agent $\hat{g}$?
Yes
No
Agent g returns information about a neighbourhood agent to the agent $\hat{g}$ $PROVIDE\_INFO(g, \hat{g})$
Agent g allocates a task at to an agent n in its neighbourhood $SINGLEALLOC(g, at, n)$
Agent g selects an action to perform using the RT-ARP algorithm
Agent g requests information from another agent n in its neighbourhood $INFO(g, n)$
Is the knowledge of agent g at its constraint limit?
No
Yes
Agent g selects an agent k from its knowledge using the SAS-KR algorithm
Agent g removes an agent from its knowledge $REMOVE\_INFO(g, k)$
Agent g adds an agent k to its neighbourhood $LINK(g, k)$
Is the neighbourhood of agent g at its constraint limit?
No
Yes
Agent g selects an agent n from its neighbourhood using the N-prune algorithm
Agent g removes agent n from its neighbourhood $REMOVE\_LINK(g, n)$

Figure 9.3: Flowchart of the ATA-RIA algorithm. *On receiving a composite task, an agent can carry out EXEC or PROVIDE_INFO actions immediately, or will choose amongst SINGLEALLOC, INFO and LINK using the RT-ARP algorithm. Taking an INFO or LINK action will lead to knowledge removal through the SAS-KR algorithm or neighbourhood pruning through the N-Prune algorithm respectively.*

Function	Definition	Summary
`sumnorm`$_p(Q)$	$Q' \leftarrow \left\{\langle a_i, \frac{p_i}{\sum_{j=1}^{N} p_j}\rangle\right\}_{\forall\langle a_i,p_i\rangle\in Q}$	*sum normalisation*, scales p values in a set $Q = \{\langle a_i, p_i\rangle\}_{i=1}^{N}$ uniformly into the range $\mathbb{R}[0,1]$, where the resulting p values sum to 1.
`softmax`$_p(Q)$	$Q' \leftarrow \left\{\langle a_i, \frac{e^{p_i}}{\sum_{j=1}^{N} e^{p_j}}\rangle\right\}_{\forall\langle a_i,p_i\rangle\in Q}$	*softmax normalisation*, converts a set $Q = \{\langle a_i, p_i\rangle\}_{i=1}^{N}$, into a probability distribution. However, softmax is a non-linear transformation where large values at the extremes are distorted to be relatively larger, making their respective actions even more likely to be selected, and smaller values smaller, making those actions even less likely to be selected.

Table 9.1: Summary of normalisation functions.

Function	Definition	Summary
`rand`(Q)	$a \xleftarrow[P(X)]{} Q,\ P(X) = \left\{\frac{1}{\vert X\vert}\right\}$	*uniform selection*, selects a value a in set $Q = \{\langle a_i\rangle\}_{i=1}^{N}$ using the uniform distribution.
`max`$_b(Q)$	$a \leftarrow \text{argmax}_b Q$	*maximum selection*, returns a value a in set $Q = \{\langle a_i, b_i\rangle\}_{i=1}^{N}$ with the maximum value of b. Randomly selects between degenerate values.
`boltzmann`$_b(Q)$	$a \xleftarrow[P(X)]{} Q,\ P(X) = \left\{\frac{e^{(b_i/\tau)}}{\sum\limits_{j=1}^{N} e^{(b_j/\tau)}}\right\}_{\forall\langle a_i,b_i\rangle\in Q}$	*Boltzmann selection*, returns a value a in set $Q = \{\langle a_i, b_i\rangle\}_{i=1}^{N}$, chosen using the Boltzmann distribution of b values with absolute temperature value τ. This means that, by altering the value τ, the transformation can increase or decrease the likelihood of selecting higher valued members of the set (and their corresponding actions) over lowered valued ones.

Table 9.2: Summary of selection functions.

The ATA-RIA algorithm

The *agent task allocation with risk-impact awareness (ATA-RIA)* algorithm (Algorithm 1) integrates the RT-ARP (Algorithm 2), SAS-KR (Algorithm 3), and N-Prune (Algorithm 4) algorithms to provide a framework for optimising task-allocation in a multi-agent system. It chooses between actions an agent can take, then updates the Q-values of each action selected using the Q-learning update algorithm[13] based on the reward values returned. We detail the steps when an agent is allocated a composite task below.

1. if there is at least one composite task allocated to the agent, and at least one atomic task it is composed from that has not yet been completed, then carry out the following steps.
2. if the agent has the capability to do so, execute an atomic task and wait for its completion [lines 3-8].
3. otherwise choose an action based on RT-ARP (Algorithm 2) [line 10].
4. carry out an action depending on the selection in the previous step;
   (a) if action selected is SINGLEALLOC:
      - allocate an atomic task [line 12].
      - wait for its completion [line 14].

   (b) if action selected is INFO:
      - request information from a neighbourhood agent [line 18].
      - then prune the knowledge base using SAS-KR (Algorithm 3) to keep within the agent's resource bounds [line 20].

   (c) if action selected is LINK:
      - add an agent from the knowledge of the agent g to its neighbourhood [line 22].
      - then prune the neighbourhood using N-Prune (Algorithm 4) to keep within agent g's resource bounds [line 23].
5. update the agent's Q-table mappings for the action taken and reward received using the *rlupdate* algorithm, therefore updating the agent's policy based the outcome of the action taken [line 26].
6. update the TSQM matrix [line 27].
7. update the action samples [line 28].
8. repeat until there are no atomic tasks that compose the composite task which have not yet been completed.

[13] See Section 9.4.1.

ALGORITHM 1: The agent task allocation with risk-impact awareness (ATA-RIA) algorithm

Input: g, an agent allocated an composite task ct.
Input: ct, a composite task allocated to agent g.
Input: W, the potential change on neighbourhoods on taking an action.
Input: $\langle K, N, SP, \Lambda, Q\rangle$, the agent state of g.
Output: $\langle K^{'}, N^{'}, SP^{'}, \Lambda^{'}, Q^{'}\rangle$, the updated agent state of g.

```
for $at \in ct$ do
    // Store current system state
    $oldstate \longleftarrow s$
    // Execute atomic task if agent has capabilities
    if $type_a(at) \in c(g)$ then
        $EXEC(g, at)$
        while $\neg complete(\{at\})$ do
            $wait(g)$
        end
        $ct \longleftarrow ct \setminus \{at\}$
    else
        // Select an action given system state
        $a \longleftarrow$ RT-ARP$(g, W, Q)$
        if $a = SINGLEALLOC(g, at, n)$ then
            $SINGLEALLOC(g, at, n)$
            while $\neg complete(\{at\})$ do
                $wait(g)$
            end
            $ct \longleftarrow ct \setminus \{at\}$
        else if $a = INFO(g, n)$ then
            // Get new agent $k$ from action
            $k \longleftarrow INFO(g, n)$
            // Add new agent to the knowledge of $g$
            $K^{''} \longleftarrow K \cup \{k\}$
            // Prune knowledge base
            $\langle K^{'}, SP^{'}, Q^{'}\rangle \longleftarrow$ SAS-KR$(g, \langle K^{''}, SP, Q\rangle)$
        else if $a = LINK(g, k)$ then
            // Add new agent to neighbourhood
            $LINK(g, k)$
            // Prune neighbourhood based on resources
            $N^{'} \longleftarrow$ N-Prune$(g, \langle N, SP\rangle)$
    end
    // Store new system state
    $newstate \longleftarrow s$
    // Update Q-value mappings using reward generated by action
    $Q^{'} \longleftarrow rlupdate(g, oldstate, newstate, a, reward(a), Q)$
    // Use the reward value to update the TSQM
    $\Lambda^{'} \longleftarrow updatetsqm(\Lambda, reward(a))$
    // Update action samples
    $SP^{'} \longleftarrow SP \cup \{(a, \phi, reward(a))\}$
end
return $\langle K^{'}, N^{'}, SP^{'}, \Lambda^{'}, Q^{'}\rangle$
```

The RT-ARP algorithm

The *reward trends for action-risks probabilities (RT-ARP)* algorithm judges the performance of an agent's current neighbourhood relative to previous ones using a TSQM. It then takes the current Q-values for an agent and transforms them through the impact transformation function. The effect is to increase the probability of an agent taking neighbourhood-altering actions, and increasing the exploration factor, when the current neighbourhood is estimated to have a lower possible locally-optimal allocation than historical neighbourhoods (See Algorithm 2).

1. select the agent's available actions and Q-values associated with the current state [line 1].
2. generate an impact transformation function from the current TSQM and use it to transform the set of action/Q-value tuples, into action/likelihood tuples [line 2].
3. sum-normalise the resulting tuples to bound the sum of their values to 1, and generate probabilities [line 3].
4. transform the exploration factor of the agent using the impact transformation function and use this for e-greedy action selection[14]. This means more exploration when recent neighbourhoods have lower quality optimal allocations achievable [lines 4-5].
5. either take the maximum-probability action or use Boltzmann selection based on the transformed exploration factor [lines 7-9].

ALGORITHM 2: The reward trends for action-risks probabilities (RT-ARP) algorithm

Input: g, an agent to select an action for.
Input: W, the potential change on neighbourhoods on taking an action.
Input: Q, the Q-table of agent g.
Output: a, the action for the agent to carry out.

```
   // Generate a set of tuples of the available actions in the agent's current state,
   // mapped to their respective Q-values in the agent's Q-table
1  $AVAIL \leftarrow \{(a, Q[s, A]) : a \in actions(g, s)\}$
   // Scale the set of tuples element-wise using impact-transformation,
   // and sum-normalise the values
2  $SCALED \leftarrow (AVAIL \circ impacttrans(W))$
3  $NORMED \leftarrow \mathtt{sumnorm}_p(SCALED)$
   // Calculate the impact exploration factor
4  $\epsilon_{\mathtt{ief}} \leftarrow impacttrans(0.5)$
   // Scale the base exploration value
5  $\epsilon \leftarrow \epsilon_{\mathtt{base}} \times \epsilon_{\mathtt{ief}}$
   // Select best action or explore with boltzmann selection
6  if $\mathtt{rand}(\mathbb{R}[0,1]) < \epsilon$ then
7      $a \leftarrow \mathtt{max}_b(NORMED)$
8  else
9      $a \leftarrow \mathtt{boltzmann}_b(NORMED)$
10 end
11 return $a$
```

The SAS-KR algorithm

The *state-action space knowledge-retention (SAS-KR)* algorithm removes learned knowledge based on action information quality [15] to the most useful information to the

[14] The exploration base constant $\epsilon_{\mathtt{base}}$ is defined in Section 9.4.5.

[15] See Subsection 9.4.2.

agent's performance is retained, and the least useful lost, while staying within the bounds of an agent's resource constraints (See Algorithm 3).

1. find non-empty state-actions in agent g's Q-table [line 1].
2. iterate through each of these states if they are now unknown to g [line 2].
3. for each of these states, check if the action information quality of the state's actions fall below the threshold value[16] [line 4].
4. remove all the action samples associated with the actions in these states [line 5].
5. remove the Q-table data associated with these states [line 6].
6. remove all knowledge of an agent if there are no actions in the stale set of actions that target that agent [lines 7].
7. if the size of the agent g's knowledge is greater than the limit $\delta_k(g)$, remove arbitrary agents from g's knowledge until its within constraints [line 10-11].

ALGORITHM 3: The state-action space knowledge-retention (SAS-KR) algorithm

Input: g, an agent whose knowledge should be managed.
Input: $\langle K, N, SP, Q\rangle$, the knowledge, neighbourhood, actions samples, and Q-table of agent g.
Output: $\langle K^{'}, SP^{'}, Q^{'}\rangle$, the updated agent state knowledge, actions samples, and Q-table of agent g.

```
   // Select all states of g in its Q-table that are non-empty
1  $S^{'} \leftarrow \{s : s \in S, Q[s, A] \neq \emptyset\}$
   // Select all currently unknown states of g that are in its Q-table
2  for $s^{'} \in unknown(g, S^{'})$ do
       // Get actions that are only available to g in unknown states
3      $UNAVAIL \leftarrow actions(g, s)$
       // Test the actions are below the information retention threshold
4      if $actval(SP, UNAVAIL, \phi) < \hat{\mu}_{\mathtt{min}}$ then
           // Remove all samples of actions in UNAVAIL
5          $SP^{'} \leftarrow SP \setminus samples(SP, UNAVAIL)$
           // Remove actions and learned Q-values
6          $Q^{'}[s', UNAVAIL] \leftarrow \emptyset$
           // Remove agents in g's knowledge that are targets of an action in an
              unavailable state
7          $K^{'} \leftarrow K \setminus \{g :\ g \in K, targets(g, s^{'}, \{g\}) \in UNAVAIL\}$
8      end
9  end
   // Check if knowledge size exceeds resource limit
10 while $|K| > \delta_k(g)$ do
       // Remove an arbitrary agent in the knowledge base but not neighbourhood
11     $K^{'} \leftarrow K \setminus \mathrm{rand}(K \setminus N)$
12 end
13 return $\langle K^{'}, SP^{'}, Q^{'}\rangle$
```

The N-Prune algorithm The *neighbourhood update (N-Prune)* algorithm ensures that the neighbourhood agents that are considered most useful to an agent's performance are the ones preferred to be retained when information must be lost due to the neighbourhood size being kept within resource constraints. Each child agent's contribution to task quality values are summed. Decay is used to reduce the relevance of older values. The infor-

[16] The uncertain information threshold constant $\hat{\mu}_{\mathtt{min}}$ is defined in Section 9.4.2.

mation on the agents with the lowest contribution is then removed (See Algorithm 4).

1. compare the neighbourhood size with the resource limits [line 1].
2. if the neighbourhood size would exceed resource constraints, and we have accumulated some quality values, then select the agent that has produced the poorest task qualities and remove it from the neighbourhood [lines 2-3].
3. if there are no quality values available then remove an arbitrary agent [line 5].

ALGORITHM 4: The neighbourhood update (N-Prune) algorithm

Input: g, an agent whose neighbourhood should be managed.
Input: $\langle N, SP \rangle$, tuple of agent state actions samples, neioghbourhood of g.
Output: $N^{'}$, updated agent state neighbourhood of agent g.

```
  // Check if neighbourhood size exceeds resource limit
1 while $|N| > \delta_n(g)$ do
2     if $|samples(SP, A)| > 0$ then
          // Find the neighbourhood agent that has returned the lowest total quality
3         $n \leftarrow minhoodval(SP, g)$
4     else
          // Choose an arbitrary neighbourhood agent
5         $n \leftarrow$ rand($N$)
6     end
      // Remove the neighbourhood agent
7     $N^{'} \leftarrow N \setminus \{n\}$
8 end
9 return $N^{'}$
```

9.6 Evaluation

Systems and algorithms

We simulated four dynamic systems to evaluate the performance of our algorithms: the stable, exploration, volatile, and large systems, with each simulation being run 100 times. Each run consisted of 100 episodes each[17], with an episode being defined by the completion of 10 composite tasks by the system (of the same composite task types in each episode).

All systems were perturbed environments, with the volatile system being more strongly perturbed than the rest. In all systems, to simulate realistic communication and environmental effects, each agent in the system had a 0.1% chance being unavailable for each episode. This value was chosen as a reasonable failure rate given the possible component failure modes of ocean-based WSN hardware, and the current and salinity effects that can disrupt communications[11], [308]–[310].

In the *stable system* we look at the performance of the ATA-RIA algorithm on the task allocation problem overall, when agents' neighbourhoods were randomly assigned on initialisation. The *exploration system* focuses on how the RT-ARP algorithm alters the probability of exploring system space to find the best neighbourhood for each agent. In this system we initialise parent agents' neighbourhoods to contain child agents

[17] An exception was made for the exploration system, which was run for 500 episodes, in order to illustrate the convergence properties of each algorithm given sufficient time to explore the system.

with atomic task qualities that are more or less than the average in the system[18]. We then investigate how agents adapt these neighbourhoods to improve performance. The *volatile system* examines the adaptability of the algorithms when the system is highly perturbed, such as during transient environmental events, by randomly making 1% of each parent agent's neighbourhood agents unavailable per-episode during a defined period of disruption between episodes 25 and 75. This represents a $10x$ increase in failure rates as compared to the stable system, a value chosen to simulate the increased component and communication failures possible in a harsh environment such as an ocean-based WSN in rough seas. Finally, in the *large system* we look at scalability, the performance of the algorithms as we increase the number of agents in the system.

Methods of analysis The solutions we develop work as a combination of multiple algorithms, each targeting an aspect of the problems in the systems we study. These involve the optimisation of task performance, resilience to perturbation, handling the loss of agent connectivity, etc., as well as taking into consideration the impact on an agent's state, neighbourhood, knowledge, and learnt values, of choosing different categories of actions. Although other algorithms were considered as comparisons, they were found to perform too slowly to complete in a practical amount of time, primarily due to the repeated selection of actions relating to agents that had become unavailable due to perturbations, and the time spent on exploration of knowledge and neighbourhood changing actions (e.g., see Section 9.7, 'Limitations of comparison'). Adding enhancement or extensions to other algorithms to cover more aspects of the problem and allow for reasonable completion times would have replicated our work into these algorithms, which would not lead to meaningful comparisons. We leave it to future research to do this integration of state-of-the-art algorithms, such as those mentioned in Chapter 5, Section 5.6, into different parts of our work to replace elements such as; Q-tables with deep learning neural networks, policy-search over Q-learning, etc., in order to develop new and possibly better performing versions of our work. For the reasons above, in this chapter's evaluation, and that of Chapters 10 and 12, we utilise theoretical baselines and sensitivity analysis, to measure algorithm performance.

Theoretical system optimal utility as a baseline comparison In a system with a single task to complete there will be an agent that can complete the task to the best or equal quality of all the available agents. With no resource sharing amongst task executions (as there are no concurrent tasks being executed by agents), enumerating the possible solutions to the allocation of tasks, AL, within the simulated systems is greatly simplified. We use this approach to give us the theoretical optimum utility in a system where atomic tasks are completed in isolation, which we then use as an easily computable comparison set of data for our simulation systems:

$$utility^*(S) = \sum_{\langle G_S(g),CL,AL,ACT,\phi\rangle \in S} allocql(atomics(AL), \emptyset) \tag{9.25}$$

In our simulations we have detailed knowledge of the state and the quality of task completion of all the agents in the system. With the removal of any resource competition these will not change with concurrency, and we can enumerate these at system initialisation to calculate $utility^*(S)$ for each allocation during each episode. We can then use the utility loss w.r.t. the theoretical optimal as a baseline comparison, as used in Figures 9.4, 9.5, 9.6, and 9.7.

[18] Child agents' atomic task qualities were set at system start time from values in the range $(0, 1]$ drawn randomly from the *normal distribution* defined by values in $X \sim \mathcal{N}(\mu, \sigma^2), \mu = 0.5, \sigma = 0.2$

Comparison algorithms

As additional comparisons, we implement two Q-learning based algorithms in the stable environment. The <qlboltz> algorithm[19] uses the reinforcement learning update strategy shown in Section 9.4.1, with the addition of Boltzmann exploration (See Table 9.2). The temperature used to reduce exploration over the lifetime of the simulation [122] was the episode count. We also used the <qlreset> algorithm for comparison, based on work extending Q-learning to non-stationary systems[305], [311], [312] as described in Section 9.4. Extending the <qlboltz> algorithm, we add a simple memory-resetting strategy that partially resets an agent's learned Q-values each episode by updating every value in the agent's Q-table with half the value's difference from the average for that state.

Configuration of algorithms

Labels for the algorithms and configurations used in the simulations are described in Tables 9.3, 9.4, 9.5, and 9.6. System parameters are included in Appendix C.1, with general and individual system values shown in Tables C.1, and C.2 respectively. The TSQM uses $(m{\times}n)$ parameters of $(3{\times}10)$[20]. The composite task frequency distribution introduced the same fixed set of tasks over a specified period, defining each *episode* of the system.

Label	Summary
<optimal>	This algorithm is used as a performance comparison as it provides the theoretical optimum system utility. Its parent agents are initialised with the most optimal neighbourhoods available in the system, and always allocate tasks to the agents that will complete them to the highest quality.
<qlboltz>	Q-learning algorithm with Boltzmann exploration using episode count temperature values.
<qlreset>	Q-learning algorithm with fixed-temperature Boltzmann exploration and episodic reset of learned Q-values.
<ataria>	The ATA-RIA algorithm.

Table 9.3: Summary of labels for the stable system. *Labels and descriptions of the optimal, qlboltz, qlreset, and ataria algorithms.*

Data presentation

Results for each system are shown in Figures 9.4, 9.5, 9.6, and 9.7. Values are shown for the percentage increase or decrease in system utility with the given algorithms in comparison to the baselines described[21]. In the stable system, the baseline is the <optimal> algorithm, <rtrap^0> in the exploration system, <nodrop> in the volatile system, and <large-optimal> in the large system. 75th percentile bands over the 100 repetitions of each simulation run are shown for Figure 9.4.

Tables of detailed results can be found in Appendix C.2 in Tables C.3, C.4, C.5, and C.6 for the stable, exploration, volatile, and large systems respectively. Statistics for each

[19] Note, we use the notation $\langle algorithm \rangle$ to denote the names of algorithms as they appear in our figures and tables of results.

[20] For a larger range of tasks, and a greater number of agents in the system, larger values of $(m \times n)$ may be preferable to allow each agent to use trends over longer-term timescales as it will have a greater range of actions it can take.

[21] After the initial use of both variance plots and p-values on the overall utility graph in Figure 9.4, only p-values are used to show the statistical significance of results shown in other graphs in order to keep those graphs easily readable.

Label	Summary
<rtrap^{0}>	ATA-RIA when the system is initialised with random neighbourhoods then explores with a constant ϵ factor, RT-ARP is disabled. This is used for a baseline comparison.
<rtrap^{+}>	ATA-RIA when the system is initialised with neighbourhoods containing 75% of the optimal neighbourhoods' agents and explores using RT-ARP.
<rtrap^{-}>	ATA-RIA when the system is initialised with neighbourhoods containing 75% of the least optimal agents and explores using RT-ARP.

Table 9.4: Summary of labels for the exploration system. *Labels and descriptions of the rtrap^{0}, rtrap^{+}, and rtrap^{-} algorithms.*

Label	Summary
<nodrop>	ATA-RIA when the system has no network instability.
<drop>	ATA-RIA when 1% of agents leave/rejoin the system each episode between episodes 25 and 75.
<nosaskr>	ATA-RIA when 1% of agents leave/rejoin the system each episode between episodes 25 and 75 but the RT-ARP and SAS-KR algorithms are disabled.

Table 9.5: Summary of labels for the volatile system. *Labels and descriptions of the nodrop, drop, and saskr algorithms.*

comparison algorithm's utility values are also shown in Appendix C.2, Table C.7. The p-values showing the statistical significance of the system utility values[22] for each simulation datasets' final episodes are shown in Table C.8.

9.7 Analysis and discussion

We now look in detail at our simulation results for each system and analyse the observed behaviours.

Stable system As seen in Figure 9.4, the <ataria> algorithm performs to 6.7% of the <optimal> algorithm after 100 episodes in the *stable system*. Initially $\sim 30\%$ of the attempted atomic task allocations made by the parent agents are not successful[23], but the failure rate rapidly falls to $< 2\%$. Although exploration is reduced as the algorithm approaches the optimal task allocation strategy, it never fully exploits the best strategy due to the effect of RT-ARP, which generates a low level of non-optimal actions. The <qlreset> performs 1.8x worse than <optimal>. Since values in agents' Q-tables are partially reset each episode, the algorithm fails to use knowledge from past experiences optimally while adapting its policy. Initially the <qlboltz> algorithm behaves similarly to <qlreset>. However, it explores and learns a policy early in the system's lifetime.

[22] These are calculated using T-tests for the null hypothesis that the expected value (mean) of a sample of independent observations is equal to the given population mean, computed using the SciPy statistics library, `scipy.stats.ttest_1samp`.

[23] Where the parent agent allocates a task at to an agent g that lacks the capability to execute tasks of that type, $type_a(at) \notin c(g)$.

Label	Summary
<large-optimal>	ATA-RIA with 10 agents, configured to give the most optimal possible RT-ARP performance in the given system.
<large-25>	ATA-RIA in a system of 25 agents
<large-50>	ATA-RIA in a system of 50 agents
<large-100>	ATA-RIA in a system of 100 agents

Table 9.6: Summary of labels for the large system. *Labels and descriptions of the large-optimal, large-25, large-50, and large-100 algorithms.*

As the system ages, the algorithm's exploration reduces, and it becomes stuck choosing actions based on the initial policy, rather than adapting to changes in the system. As a result, it reaches 2.3x of optimal performance by episode 25, but then worsens to 3.0x by episode 100 as the difference between its stationary policy and the newer, more optimal ones increases[24].

Overall, these results show that the <ataria> algorithm can optimise system utility well in a stable system. Although the effect of RT-ARP means that ATA-RIA is not fully optimal under these conditions, it also improves its ability to adapt to changes as the environment becomes more dynamic.

Limitations of comparisons

The average time taken for each algorithm to complete an episode in the stable system as the parent agent count increases is shown in Table 9.7. As the *large system* involves more parent agents, and the *volatile system* is increasingly non-stationary, the episode times involved for the comparison algorithms proved intractable for useful simulation runs. Due to this, we only simulate the ATA-RIA algorithm for the systems that follow. For further discussion see Appendix B, Section B.2.

Algorithm	Average time per-episode by parent agent count (secs)				
	1 agent	2 agents	3 agents	5 agents	10 agents
<ataria>	0.7	1.6	2.9	3.7	6.4
<qlboltz>	1.1	19.6	162.9	223.7	2302.0
<qlreset>	1.2	16.7	141.8	171.5	1862.2

Table 9.7: Runtimes of algorithms in the stable system. *Data shows the average time taken to complete episodes in each of our target systems. The comparison was carried out using a AMD Ryzen 9 3900X 12-Core Processor, 3793 Mhz, 12 Core(s), 24 Logical Processor(s) Processor with NVIDIA GeForce RTX 2070 SUPER GPU acceleration.*

Exploration system

Next we examine the exploration of state-space in the *exploration system*, in Figure 9.5. The <rtrap^{+}> algorithm gains a 67.0% improvement in system utility compared to <rtrap^{0}> after 500 episodes. <rtrap^{-}> improves 62.7% in task completion performance, with the expectation that this would merge with the utility levels of <rtrap^{+}> given more episodes. The RT-ARP algorithm acts of a proxy comparison of the current

[24] Due to the dynamic agent composition of the system, and other parent agent changing their allocation strategies over time.

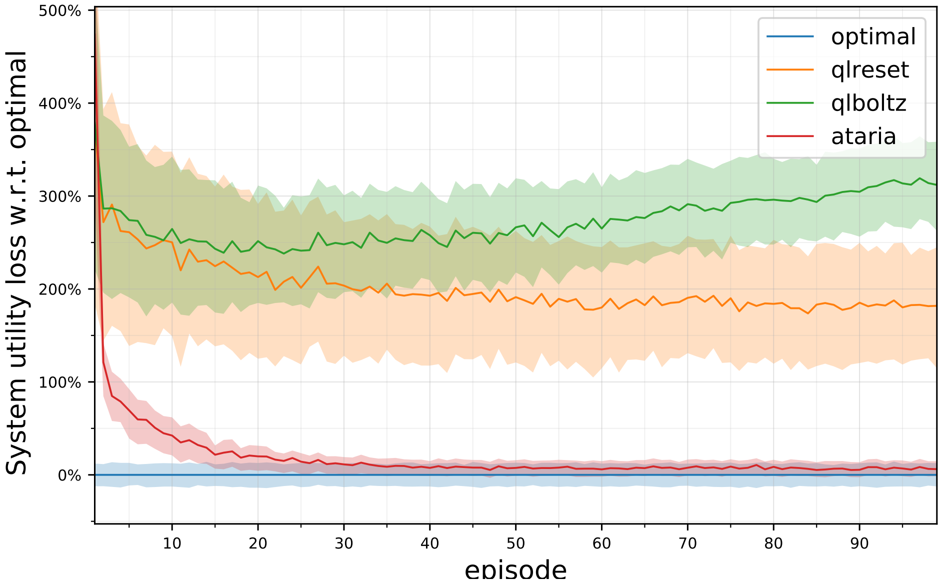


Figure 9.4: *System utility comparison to the system optimal in the stable system, as measured by the percentage difference in utility between the specified algorithms and the* <optimal> *baseline algorithm.*

allocation quality for an agent, to the locally-optimal allocation, and system-optimal allocation qualities for that agent. It drives the agent into better neighbourhoods for its task allocations and increases the system's utility. As the current neighbourhood nears the optimal neighbourhood for that agent and its tasks, the rate of exploration falls.

Volatile system In the *volatile system* in Figure 9.6 we see the SAS-KR algorithm's effect on system resilience and recovery. Before the disruption to agent connectivity is introduced at episode 25, the algorithms' performances are equivalent. On introducing instability, the performance of the <drop> and <nosaskr> algorithms deteriorate by 72.5%, gradually improving to 59.7% over the course of the disruption. After instability stops at episode 75 <drop> recovers to 9.7% of the performance of the non-impacted <nodrop> algorithm by episode 100, as compared to 54.6% for <nosaskr>.

As the SAS-KR algorithm retains the most up-to-date, and least uncertain actions and associated Q-values, better information about past actions and neighbourhoods is kept by the agent as compared to with it disabled. When the instability is removed, the quality of knowledge kept by the <drop> algorithm is higher than in <nosaskr>, allowing a quicker recovery to more optimal neighbourhood formations, and so task-allocation quality and overall system utility.

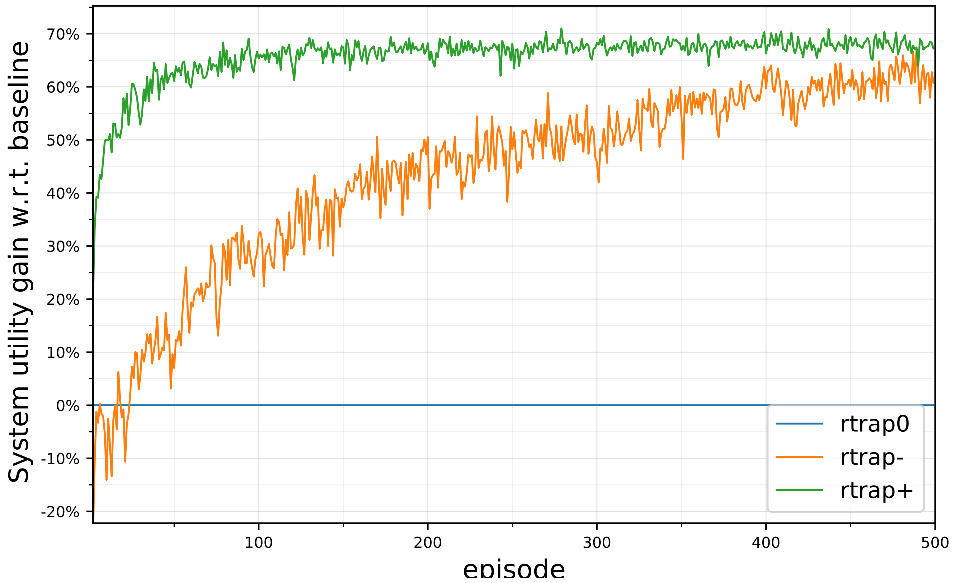


Figure 9.5: *System utility comparison to the system optimal in the exploration system, as measured by the percentage difference in utility between the specified algorithms and the <*`rtrap`^{0}*> baseline algorithm.*

Learning under uncertainty and disruption

In the stable system, this ability of SAS-KR algorithm to retain higher quality knowledge helps guide exploration. When the environment is more disrupted however, it has a greater effect as agents' knowledge changes more rapidly. The RT-ARP algorithm increases exploration during the early episodes when there is large uncertainty in which action choices are optimal. This enables the agent to learn quickly, and slow down learning as performance improves. Similarly, it will increase exploration during disruptions as the performance of agents in these environments is most likely less rewarding than in the past. If the disruption is transient, the agent can quickly re-apply its retained knowledge to recover performance. There may be improvements possible in how well these algorithms perform with further research, however, how the quality of knowledge is judged, and how aggressively the RT-ARP algorithm moves between exploration and exploitation, is dependent on the desired behaviour of the specific multi-agent system they are applied in.

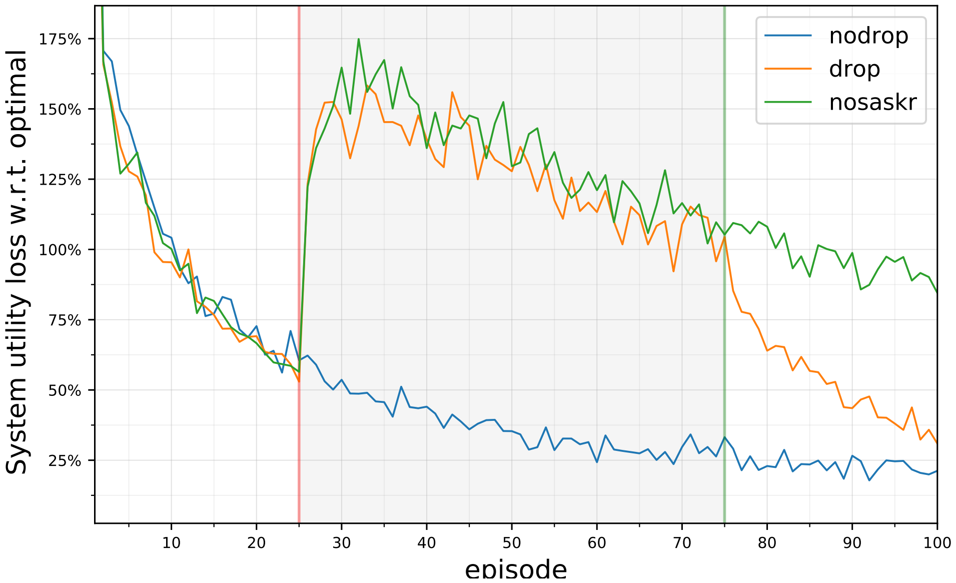


Figure 9.6: *System utility comparison to the system optimal in the volatile system, as measured by the percentage difference in utility between the specified algorithms and the* <nodrop> *baseline algorithm.*

Large system The *large system* is shown in Figure 9.7. Here we see the <large-25> algorithm perform within 3.6% of the <large-optimal> algorithm, the optimal performance possible for the ATA-RIA algorithm in the system. The <large-50> and <large-100> algorithms optimise system utility to within 7.2% and 8.6% of <large-optimal> by the completion of 100 episodes. As expected, the system utility of the ATA-RIA algorithm is initially poorer with increasing number of agents in the system. On initialisation of the system, there is a greater likelihood of parent agents being in neighbourhoods with agents that have lower than average atomic task qualities available. There is also a larger system space for the algorithm to search. Even so, the ATA-RIA algorithm shows good performance in optimising the system utility to within 10% of optimal with a system of 100 agents.

Although there is a more rapid improvement in utility with fewer agents since the system-space to learn is smaller, further investigation shows that the performance of the three systems converges with increasing episodes. However, due to the compute and storage limitations of running the simulations repeatedly for longer periods, we have limited the comparisons to 100 episodes. Further research on more powerful simulation platforms would be expected to show a similar behaviour and convergence properties for larger systems.

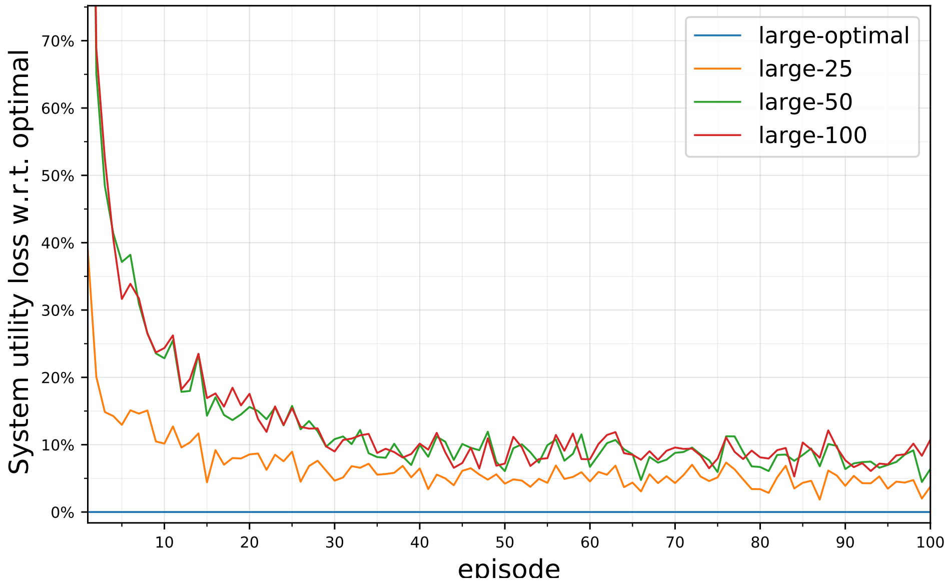


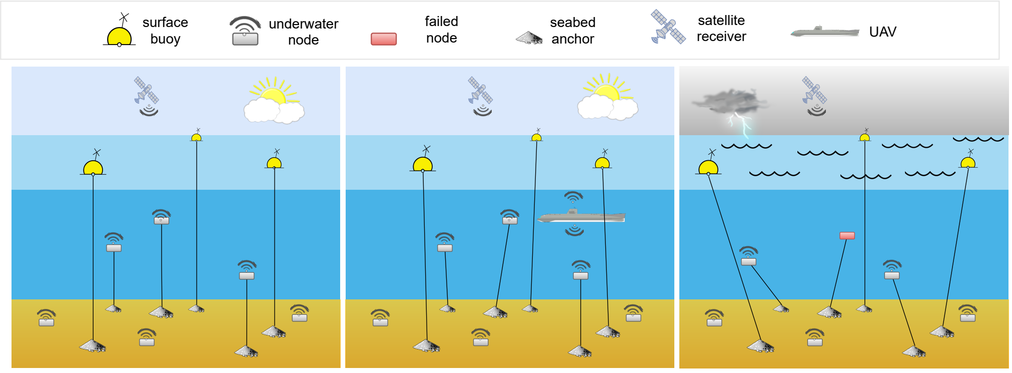
Figure 9.7: *System utility comparison to the system optimal in the large system, as measured by the percentage difference in utility between the specified algorithms and the* `<large-optimal>` *baseline algorithm.*

Summary

Overall, the evaluation of the algorithms presented shows that they perform well at task allocation in both stable and unstable environments, as well as scaling to larger systems. The ATA-RIA algorithm improved system utility to 6.7% of the optimal in the simulated system. The RT-ARP algorithm reduced exploration as the system utility approached optimal, and adapted well in response to disruption. It allowed agents to alter their neighbourhoods from areas of state-action space that would not allow task completion to those where it would be possible. In environments with disrupted connectivity, the retention of learned knowledge through SAS-KR allowed for quicker re-optimisation and adaptation of neighbourhoods, over 5× better than when RT-ARP and SAS-KR were disabled, and there was no adaptive exploration or knowledge retention strategy.

9.8 Discussion of the applicability of algorithms to an ocean-based WSN

We now describe a system inspired by a real-world use case to which our work could be applied, relating the behaviour of our algorithms experimentally to the challenges presented. Previously, we have briefly given examples of realistic WSN systems (See examples 8.2.1, 8.5.1 in Chapter 8 and Example 9.4.1 in this chapter). These examples have been chosen to illustrate the theoretical concepts discussed, the challenges, and key properties of dynamic multi-agent systems. They require ad-hoc learning of agent neighbourhoods. Agents enter and leave the system through component failure and re-deployments. The capabilities of different agents to complete tasks and the qual-



(a) Deployment and initial ad-hoc network

(b) Mild currents and mobile UAV

(c) Storm disruption and node loss

Figure 9.8: An illustration of a common ocean-based WSN system deployment. *In Figure 9.8a, the nodes are deployed and an initial task optimisation is learned. In Figure 9.8b, the ATA-RIA adapts the actions of nodes to account for movement due to currents, and passing UAVs. In Figure 9.8c, nodes are highly disrupted and some fail. The SAS-KR and RT-ARP algorithms work to quickly re-establish an optimal configuration from past knowledge and the prioritisation of exploration as the environment stabilises.*

ities they can complete them to can vary due to placement, obstructions, different instrumentation, and wear and tear of components. Our work focuses on adapting to this changeability in the optimal allocation of tasks within a system. However, these examples are also relevant to highlight the types of existing, real-world systems that our work could be applied to, and verified against.

We focus on a sub-category of these systems that contain complex deployments of sensor nodes for ocean monitoring, often described as the Internet of Underwater Things, or the Ocean of Things[8], [11], [255], [313], [314]. The networks built for these systems are termed Underwater Wireless Sensor Networks (UWSN)[315], [316].

Challenges in underwater wireless sensor networks

Figure 9.8 shows a common UWSN scenario[256], [317]–[320] where the deployed nodes can be tethered buoys or submerged sensors, as well as mobile Unmanned Autonomous Vehicles (UAV). In the system shown, all the nodes form ad-hoc communication groups to carry out ocean monitoring tasks such as temperature and salinity measurement[321]. Since radio transmissions are absorbed quickly underwater, acoustic transmission is used. This change affects key properties of the UWSN[318], [322] that our multi-agent algorithms must adapt to if they are to remain useful.

- Acoustic signals have a much lower bandwidth, 10kbps compared to 250kbps for radio transmission.
- Signals propagate over 100,000 times slower than radio.
- Signals travel much further, ~10km compared to 100m for radio.
- The nodes' transmission components are hard to recharge, so good power efficiency is essential.
- Underwater links are more unreliable due to corrosion, variable salinity density, and absorption of signals through water. Bit rate errors can be high, and connectivity intermittent.

- Nodes can move under currents, often severely during rough seas, affecting the optimal network configuration.

Initial deployment

The nodes need to form an ad-hoc network to receive task requests and return monitoring data from their sensors (Figure 9.8a). The ATA-RIA algorithm enables nodes to discover each other, establish the capabilities of other nodes, and form groups to optimise their ocean monitoring tasks. Energy resource usage can form part of the quality metric of task completion, encouraging the agents to learn better policies for power efficiency. RT-ARP will ensure that they prioritise discovery at this initial stage, and move towards the efficient completion of temperature and salinity sensing tasks as nodes learn more about the system. In our simulations this is the behaviour shown in Figure 9.4 up to approximately episode 25.

Behaviour in calm weather

In stable conditions there will still be some movement of nodes due to currents, replacement of buoys and sensors due to wear, and intermittent mobile nodes such as UAVs (Figure 9.8b). The RT-ARP algorithm maintains a small amount of exploration to not only optimise communication between nodes that have established a neighbourhood group, but also cautiously discover other nodes to add to the group. The SAS-KR algorithm helps with transient impacts such as high salinity corrupting node-to-node communications, or UAVs moving in and out of the system. It does so by allowing nodes to retain selective knowledge between last known sightings, so that when these nodes or UAVs reappear, agents can recall that knowledge and quickly re-adjust. In Figure 9.6 in Section 9.7 we see how the RT-ARP and SAS-KR algorithms achieve this. They help adapt to the intermittent loss of nodes during the period between episodes 25 and 75 by continuing to optimise agents' task completions during that period.

The N-Prune algorithm makes sure that throughout the discovery process and adaptation, a node's limited resources are not overstretched, and the least useful other nodes known to it are removed from its memory to keep within its resource constraints.

Behaviour in rough seas

During instability, the effects seen in calm weather are magnified (Figure 9.8c). In this situation, current reward trends are likely less favourable than those in the past, during calmer conditions. This pushes the RT-ARP algorithm towards more extreme exploration as nodes are lost, destroyed, or displaced. As the storm passes and the position of nodes, currents, and salt density stabilises, SAS-KR allows the nodes to remember previous nodes it may have lost contact with, or whose capabilities had been disrupted. As RT-ARP will accelerate exploration until the system's performance is comparable to historical values, the use of existing knowledge increases the speed of this recovery by removing the need for nodes to re-learn everything they learned about other nodes prior to disruption. For example, the period of the storm would be similar to the episodes $25 - 75$ in Figure 9.6 in our simulation, with the ocean calming after 75 episodes. The behaviour of the RT-ARP and SAS-KR algorithms in accelerating recovery in the simulation should be equivalent to our UWSN system after a disruptive storm.

9.9 Summary

As we have shown in this chapter, with the ATA-RIA algorithm optimising agents' task allocations, RT-ARP adapting exploration based on reward trends, and the SAS-KR and N-Prune algorithms managing knowledge and neighbourhood retention respectively, the contributions presented here combine to optimise task-allocation in multi-agent systems. The results of our evaluation show that the combined algorithms give good task allocation performance compared to the theoretical optimal available in the simulated systems, and are resilient to system change with constrained computational cost and other resource usage. This indicates a good basis for successful application to real-life systems where there are resource constraints, and dynamic environments.

Chapter 10

Resource allocation

This chapter examines how agents can allocate their available resources in order to optimise the completion of the incoming tasks being allocated to them by other agents. We evaluate the performance of our developed solution using a uniform resource allocation baseline alongside sensitivity analysis (Contribution 2).

10.1 Introduction

The allocation of resources by an agent to improve its performance in completing certain tasks over other ones is a key element of problem domains such as; information exchange and coordination amongst autonomous vehicles[3], and the allocation of shared resources in cloud computing[60]. The challenge in developing efficient and robust algorithms stems from the dynamic nature of these systems, with many components communicating and interacting in complex ways. The algorithm we develop in this chapter tackles this problem by;

- estimating the demand for an agent's resources to complete tasks allocated by groups of other agents;
- combining these multiple per-group estimates into an aggregated estimate of the best allocation of its resources to meet the needs of all these groups of agents;
- in doing so, the agent learns to allocate its resources to complete atomic tasks in such a way as to maximise the utility of the system as a whole.

Our algorithm is designed to overcome the issues of other approaches described in Chapter 4 Section 4.4 such as lack of scalability, complex resource allocation side effects, and the trade-off between joint and independent action learning. This solution is useful where;

- there are competing demands for shared resources in a MAS;
- agents can prioritise amongst the tasks allocated to them;
- the environment is dynamic, for instance, agents join and leave the system;
- the distribution pattern of the allocation of atomic tasks to the agent allocating

its resources changes over time.

Chapter structure The rest of the chapter will introduce the resource allocation problem, defining resources and their allocation in Section 10.2. We build on the work from Chapter 9 in this section to include resource usage as a factor in atomic task completion, and examine how this affects the quality of composite tasks and overall system utility. In summary, we;

- extend the system definition in Definition 8.2.1 to include types of resource;
- extend the system state from Definition 8.3.5 to include the availability of resources to agents, and their allocation of those resources amongst atomic task types;
- extend the atomic task quality in Definition 9.3.1 to include the effect of an agent's allocation of resources;
- define the quality of a composite task, and how we can derive values from it as a reward signal for agents that complete corresponding atomic task;
- define system utility using composite task qualities, rather than the sum of atomic task qualities as previously in Chapter 9.

We develop intuition for our strategy for resource allocation optimisation in Section 10.3, including our method of grouping and modelling agents for learning in Subsections 10.3.1 and 10.3.2. Following that work, we extend our definition of agent state to include these new elements in Section 10.3.3. We formally lay out our algorithms in Section 10.4 before evaluating their performance through simulation and analysis in Section 10.5.

We formulate the resource allocation problem in a generally applicable way so that it can be applied to multiple domain-specific use cases without modification, as well as being easily integrated with the previous task allocation work in Chapter 9 in subsequent chapters.

10.2 Resource allocation in a multi-agent system

Resources We define a resource $res \in \mathcal{RES}$ as a quantity of one of the possible *resource types*, $\mathcal{RP}$, in the system, where $type_r : \mathcal{RES} \rightarrow \mathcal{RP}$ maps the domain of resources to resource types. In this work we define resources with common characteristics such as being;

- *isolated*, a resource is dedicated to a single agent and not accessible to other agents in the system (i.e. one agent cannot utilise energy from another agent's battery);
- *statically assigned*, resources cannot be moved or reallocated to other agents;
- *non-differentiable*, resources of given resource type are indistinguishable by any other characteristics beyond quantity (i.e. if two quantities of a resource are equal, they will make an identical contribution to atomic task quality);
- *divisible*, an agent with an available resource can split the usage of that resource between multiple tasks.

In defining our solution to the resource allocation problem we make no assumptions about whether the resources described are finite and will therefore run out, or infinite in supply.

Example 10.2.1 (*Energy resource in a WSN system*). A WSN system is deployed in the environment surrounding a nuclear waste store. Each node in the system carries a sensor to measure radioactive contamination, powered by a battery that is regularly recharged by a solar panel attached to the node. In this case, the resource to be allocated by the agent in order to complete tasks is energy from the battery. This energy resource cannot be accessed or delivered to another node in the system as nodes are isolated apart from wireless communications and not part of a connected power grid. The energy resource does not run out (as long as there is sufficient time and sunlight for the battery to recharge more quickly than it is used up). Each agent's energy is identical to that of every other agent, and achieves the same amount of work by their sensors (ignoring other effects such as manufacturing variability, and differing levels of component degradation amongst agents).

The unknown value of resource allocations

The aim of individual agents in the system is to ensure those atomic tasks that are of most value to a composite task are performed to the highest quality, thus ensuring the composite tasks produce the best results. For child agents enacting atomic tasks, this means allocating more resources to high value tasks.

There are obstacles to this goal. First, the value of the atomic task to the composite task may not be known in advance, as said above. Second, a given child agent cannot know when it will receive atomic tasks to enact because, even if the frequency by which composite tasks are received by the system is constant and known, it is a choice of a parent agent as to who to allocate atomic tasks to in any given instance. If a child agent has already allocated resources to one atomic task under execution, then it cannot use those resources for a new task of potentially higher value.

To address this latter point, we assume that each child agent allocates in advance a portion of its resources to each atomic task type. We are agnostic whether tasks are then executed sequentially or in parallel, which will vary per application. An agent's resource allocation can change over time: specifically, the solutions presented later in this chapter allow an agent to learn a good resource allocation for the tasks it is assigned and the value they have to composite tasks. Which child agents will carry out the atomic tasks of a composite task are determined by the task allocation of the parent agent enacting the composite task.

Resource availability

Each agent has a varying amount of each resource type available for it to allocate to completing its tasks (e.g. available battery power that changes over time with usage). We denote all these possible *resource availabilities* for agents in the system, $\mathcal{RAV} = 2^{(\mathcal{G}\times\mathcal{RES})}$. How much resource of a given type rp an agent has available in a resource availability RAV is given by $resavailable\colon \mathcal{G}\times\mathcal{RP}\times\mathcal{RAV}\to\mathcal{RES}$.

Resource allocation An agent g can share its available resources amongst each type of atomic task it could perform, giving a set of tuples $\langle g, ap, res\rangle$ of each atomic task type ap and the quantity of each resource type, res allocated to completing tasks of that type. The set of these tuples for a set of agents G, defines the set of *resource allocations*, $\mathcal{RAL} = 2^{(\mathcal{G}\times\mathcal{AP}\times\mathcal{RES})}$. The resources allocated to an agent g, for task type ap, by a resource allocation RAL are given by $resallocation\colon \mathcal{G}\times\mathcal{AP}\times\mathcal{RAL}\to\mathcal{RES}$.

System definition We extend our system from Definition 8.2.1 to include the use of the resources types that will be used in the completion of atomic tasks, and also add resource availability and allocation to the system state from Definition 8.3.5.

Definition 10.2.1 (*System (resource allocation)*)**.** The distributed task-allocation system (DTAS) is defined by a tuple $\langle AT, CT, A, G, RP\rangle$, where RP is a set of resource types needed to perform tasks.

Definition 10.2.2 (*System State (resource allocation)*)**.** Given a DTAS we define its state as a tuple $S = \langle G_S, CL, AL, ACT, RAV, RAL, \phi\rangle$ where RAV is the set of resource availabilities, and RAL the set of resource allocations, in the system.

Example 10.2.2 (*Resource availability and allocation in a radioactive contamination monitoring WSN*)**.** Two agents are deployed in an environment to monitor radioactive contamination, $G = \{g_1, g_2\}$. The agents can perform tasks to take radiation measurements, or to sample oxygen quality, $AP = \{ap_{rad}, ap_{oxy}\}$. Each task requires the use of an energy resource type, $RP = \{rp_{pow}\}$, which is available to an agent solely from its own battery. Agent g_1 has a quantity of energy available $res_1 : type_r(res_1) = rp_{pow}$, and g_2 has quantity $res_2 : type_r(res_2) = rp_{pow}$. Using our definition of resource availability previously:

$$RAV = \{\langle g_1, res_1\rangle, \langle g_2, res_2\rangle\}$$

$$resavailable(g_1, rp_{pow}, RAV) = res_1$$
$$resavailable(g_2, rp_{pow}, RAV) = res_2$$

Agent g_1 allocates half of its available energy resource res_1 to each of the two task types it can perform. Whereas agent g_2 dedicates all of its available energy resource res_2 to radiation sensing tasks, giving the resource allocations:

$$RAL = \{\langle g_1, ap_{rad}, res_1/2\rangle, \langle g_1, ap_{oxy}, res_1/2\rangle, \langle g_2, ap_{rad}, res_1\rangle, \langle g_2, ap_{oxy}, \emptyset\rangle\}$$

$$resallocation(g_1, ap_{rad}, RAL) = \{res_1/2\}$$
$$resallocation(g_1, ap_{oxy}, RAL) = \{res_1/2\}$$

$$\begin{aligned} resallocation(g_2, ap_{rad}, RAL) &= \{res_2\} \\ resallocation(g_2, ap_{oxy}, RAL) &= \emptyset \end{aligned}$$

Atomic task quality (extended)

Given our definition of resources and their allocation by agents, we can now extend our definition of atomic task and allocation quality given in Definitions 9.3.1 and 9.3.2.

Definition 10.2.3 (*Atomic task quality (extended)*)**.** The *atomic task quality* for an agent completing a task depends on the type of the task, the agent's concurrent atomic allocations, and the amount of each resource it has allocated to completing that type of task:

$$atomicql\colon \mathcal{AP} \times \mathbb{N}_0 \times 2^{\mathcal{RES}} \to \mathbb{R} \tag{10.1}$$

As well as the quality of atomic task completions, there may be domain-specific contributions to allocation quality, dependent on the requirements of each specific system. E.g. to better distribute resource usage amongst agents when completing a composite task.

Domain-specific allocation quality components

To provide a generalised formulation of allocation quality we introduce two elements into our definition of allocation quality from Chapter 9, Definition 9.3.2; a *domain-specific atomic quality* component, $domtask\colon \mathcal{AT} \times \mathcal{G} \to \mathbb{R}$, which scales the atomic task quality; and a *domain-specific allocation quality* component, $domql\colon \mathcal{AT} \times \mathcal{G} \to \mathbb{R}$, which extends our allocation quality definition to include other factors.

Atomic allocation quality (extended)

Definition 10.2.4 (*Atomic allocation quality (extended)*)**.** The *atomic allocation quality* of a set of atomic tasks AT, their atomic task allocation AL, and resource allocation RAL is given by $allocql\colon 2^{\mathcal{AT}} \times \mathcal{AL} \times \mathcal{RAL} \to \mathbb{R}$, such that:

$$allocql(AT, AL, RAL) = \sum_{\forall at \in AT, \exists \langle at, g, \widehat{g} \rangle \in AL} \begin{array}{r} atomicql(type_a(at), tnum, RES').domtask(at, g) \\ + domql(at, g) \end{array} \tag{10.2}$$

where $tnum = |concurrent(AL, g)|$ and $RES' = resallocation(g, type_a(at), RAL)$.

In this chapter we will focus purely on the resource allocation problem and assume;

1. the quality of atomic task completion is independent of task concurrency, $|concurrent(AL, g)| = 1$;
2. all agents complete tasks to the same quality if they use the same amounts of the same resources;
3. there are no domain-specific contributions to allocation quality, $domtask(at, g) = 1$ and $domql(at, g) = 0$.

Under these assumptions, in this chapter Equation 10.2 is simplified to:

$$allocql(AT, AL, RAL) = \sum_{\forall at \in AT, \exists \langle at, g, \widehat{g} \rangle \in AL} atomicql(type_a(at), 1, RES')$$

Note that assumptions 1 and 2 above are to simplify this chapter's work only, we remove these assumptions and combine our work both task and resource allocation in Chapter 11. Assumption 3 is only in place until our algorithms are applied to specific domains, as in our WSN case study in Chapter 12.

Component tasks proportional value

Each atomic task in a composite task will give some value to the outcome of that composite task, corresponding to how significantly it contributed relative to other atomic tasks. This is not the same as result quality, e.g. an atomic task may have been performed to exceptionally high quality but produced results that were replicated by other successful tasks, whereas another task performed at lower quality may have produced an important and distinct product/finding. In general, the value of each atomic task to a composite task may be unknown until the composite task is completed.

Definition 10.2.5 (*Component tasks proportional value*)**.** The *proportional value* of each component atomic task of a composite task ct is expressed by a mapping from the atomic tasks comprising ct to the fractional value each atomic task contributed to ct as judged after ct's completion, $propval\colon \mathcal{AT} \times \mathcal{CT} \to \mathbb{R}$, where $\sum_{at \in ct} propval(at, ct) = 1$ and the value of an atomic task to a composite task of which it was not part is zero.

Composite task quality The results of atomic tasks are returned to the parent agent enacting their encapsulating composite task. The resulting qualities of these tasks are determined from the corresponding task and resource allocations (using the quality function). The quality of result of a composite task depends on the qualities of its component atomic tasks and the value of each to the composite task, i.e. it is higher where the most important tasks were performed well.

Definition 10.2.6 (*Composite task quality*)**.** The *composite task quality* of a set of composite task CT, performed under atomic task allocation AL and resource allocation RAL is given by $compositeql\colon 2^{\mathcal{CT}} \times \mathcal{AL} \times \mathcal{RAL} \to \mathbb{R}$, such that;

$$compositeql(CT, AL, RAL) = \sum_{\forall ct \in CT} \sum_{\forall at \in ct} allocql(\{at\}, AL, RAL).propval(at, ct) \tag{10.3}$$

Absolute value of a component task From this we can define, first, the absolute value of each atomic task to the system being the product of the composite task's quality and the atomic task's relative value in generating that quality and, second, the utility of the system during a time period as being the total qualities of the composite tasks it executes in that period.

Definition 10.2.7 (*Absolute value of a component task*)**.** The *absolute value* of each component atomic task of a composite task ct executed under atomic task allocation AL and resource allocations RAL is expressed by a mapping from the atomic tasks comprising ct to the actual value provided by that task according to the quality of the composite task and the proportional value of the atomic task within the composite one, $absval\colon \mathcal{CT} \times \mathcal{AT} \times \mathcal{AL} \times \mathcal{RAL} \to \mathbb{R}$, so that;

$$absval(ct, at, AL, RAL) = compositeql(\{ct\}, AL, RAL).propval(at, ct) \tag{10.4}$$

Child agents can use the absolute value of component tasks to learn the value of their actions to the completion of the composite task of a parent agent.

Example 10.2.3 (*The absolute value of a component task in a V2X system*)**.** Vehicle A is moving within a vehicle-to-everything (V2X) communication system with another vehicle B. Each vehicle can complete atomic tasks to provide their position, tasks of type ap_{pos}, and local congestion information, of task type ap_{con}, to other vehicles. Both types of atomic task require use of an agent's energy resource res_{pow}. As it has a fixed capacity, if more energy is allocated to tasks of type ap_{pos}, then the vehicle can provide more frequent and accurate positional updates to other vehicles, but less is allocated to ap_{con} tasks, and traffic congestion information is less regular and less detailed.

> Vehicle B has composite task $ct = \{at_{pos}, at_{con}\}$ and allocates both of the component atomic tasks to vehicle A. Vehicle B is moving away from vehicle A, and so may be affected more by congestion than the location of vehicle B, so it values congestion information more than positional updates from vehicle A, which are less relevant. In other words, $propval(at_{con}, ct) > propval(at_{pos}, ct)$. Therefore, vehicle A can increase the overall sum of component task values of vehicle B's composite task ct by allocating proportionally more of its res_{pow} to ap_{con} tasks. This will increase the composite task quality of ct, and so the utility of the system overall.

System utility

In Chapter 9 we assumed that there was no interdependence between the value of different atomic tasks composing a composite task, and so we could use the simple sum of atomic task qualities to judge the utility of the system. By including a parent agent's judgement of the relative value of atomic tasks composing its completed composite tasks, we now need to use the composite task quality to measure the utility of the system.

Definition 10.2.8 (*System utility (extended)*)**.** The *utility* of a system is then the sum of composite allocation qualities of each allocation in a set of system states, $utility\colon 2^{\mathcal{S}} \to \mathbb{R}$ so that:

$$utility(S) = \sum_{\langle G,CL,AL,ACT,RAL,RAV,\phi\rangle \in S} compositeql(composites(CL), AL, RAL) \quad (10.5)$$

10.3 Learning high value resource allocations

The utility of the system can be improved by child agents preferring to allocate their resources to completing atomic tasks that will be of more value to the corresponding composite tasks of parent agents. A child agent allocates resources to atomic task types in advance of being allocated tasks of those types to complete due to the unpredictable nature of the allocation from its perspective. Our solution is influenced by three problems arising from the problem domain.

Unknown task value

The first problem is that the value of an atomic task to its corresponding composite task is unknown in advance to the agent it is allocated to. We assume that, while the value of may not be known, there may be biases or patterns to be learnt. We further assume that these patterns will correspond to the parent agents from whom atomic tasks are allocated, i.e. a given parent agent will typically be performing composite tasks for a given purpose within the system, the relative value of each atomic task may have similarities across the composite tasks allocated by that parent agent. Much of the solution below would apply equally should a particular system have another way to categorise incoming tasks, other than the parent agent, relevant to learning their likely value.

Varying task allocation frequencies

The second problem is that a parent agent will be making atomic task allocation decisions that change which atomic tasks a given child agent receives, and the parent agent's strategy could vary over time. Additionally, the frequencies at which composite tasks of each type arrive in the system may change. This means that a child agent will not consistently receive atomic tasks of different types at the same rate and their arrival will not be uniform. As a child agent can only have one resource allocation, it will not be beneficial to allocate many resources to a task type for which tasks of that type are likely to be high value if arriving from a given parent agent, but that parent agent is not allocating the child any tasks at this time.

Learning resource cost

The final problem is that learning requires resources and child agents have limited resources, as specified in the previous section. A system may be large in the number of parent agents, therefore it may not be feasible for a child agent to learn patterns of which atomic tasks types are most valuable for every parent agent due to its memory or computational limitations.

Learning adaptive resource weightings

The solution we propose uses reinforcement learning to adapt weightings that measure the value of each type of resource when allocated to the completion of each atomic task type, to the value of the atomic tasks the agent receives from parent agents. We take the approach that;

1. each child agent will allocate an amount of resources to learning parent agent task value patterns. In this section, the resource availability will be fixed in advance. In our work in Chapter 12 we allow the resource availability to vary, simulating the use of energy from batteries in a WSN system[1];
2. a child agent groups parent agents in the system and treats each group as if it was one parent agent for the purposes of learning. The number of agents per group is dictated by the resources allocated to learning: more resources allows for an increased number of smaller groups;
3. for each parent agent group there is a learnt model of the value of each atomic task type. When a parent agent completes a composite task, the child agents that completed the corresponding atomic tasks are sent the absolute values for those tasks. The child agent then adjusts its model for the relevant parent agent group;
4. the child agent regularly aggregates the learnt models to determine a resource weighting across atomic task types and adjusts its resource allocation accordingly for subsequent tasks.

10.3.1 Modelling the resource allocation for groups of agents

As learning requires the use of limited resources, a solution that models the value of resource allocation for each parent agent individually would be limited in scalability, as the resources required for learning increase with the number of parent agents. In contrast, if the solution used the incoming atomic tasks for all parent agents and modelled them as-one, it would be unlikely to provide optimal results where there were many agents making requests, due to the problems of unknown atomic task values, varying allocation frequencies, and learning resource cost as described previously in Section 10.3.

Parent groups

To mitigate these problems we combine parent agents into multiple groups. Tasks allocated from each group of parent agents are treated as the same distribution for resource allocation modelling by the child agent. Resource weightings for atomic task types are similarly defined for each parent agent group rather than for each parent

[1] We leave varying the balance of resources allocated to learning versus task execution to future work. When referring to the resources possessed by an agent, we will exclude the resources allocated to learning for simplicity

agent individually. This approach allows us to constrain the usage of resources for modelling by the child agent, while retaining information on parent agents' requests over longer time periods.

Definition 10.3.1 (*Parent group*). A *parent group* is defined as a fixed mapping for each child agent to sets of parent agents that are grouped together for atomic task value modelling, $group\colon \mathcal{G} \times 2^{\mathcal{G}} \to 2^{\mathcal{G}}$

Figure 10.1 illustrates how the distribution of task types amongst two parent groups affects the allocation of resources.

Example 10.3.1 (*Information loss with time*). A parent agent pg_1 allocates an atomic task at_1 to a child agent cg at time ϕ_1, which obtains the best results given a certain resource allocation by cg. This atomic task is followed by atomic task allocations from many other parent agents. After a period of time, pg_1 allocates another task at_2 of the same type $type_a(at_2) = type_a(at_1)$ at time ϕ_2. If all parent agents were modelled as one group, since there would be many changes to the model between each request by pg_1, the resource allocation strategy of cg at ϕ_2 is unlikely to be close to optimal for the demands of pg_1. This causes a loss of knowledge between ϕ_1 and ϕ_2 of the weightings that would give the best atomic task value for atomic task types $type_a(at_2)$, when allocated by the parent agent pg_1. By using smaller groups of agents for modelling, the changes in the model caused by other parent agent task allocations can be lessened between the similar task allocations from pg_1 at ϕ_1 and ϕ_2.

Parent group weights

We then assign weights to each parent group and resource type to represent the learnt value of allocating each resource type to the atomic tasks arriving from that group.

Definition 10.3.2 (*Parent group task weights*). For each parent group of an agent g, there is a set of *parent group weights* for each type of resource rp, representing the estimated value of allocating resources of that type to each type of atomic task, to that group of agents, $weight\colon \mathcal{G} \times 2^{\mathcal{G}} \times \mathcal{AP} \times \mathcal{RP} \to \mathbb{R}$. Where, given an agent can accept tasks in the set AP, $\sum\limits_{ap \in AP} weight(g, G, ap, rp) = 1$.

Parent group weights matrix

To simplify some future equations for the blending and combination of weight values we combine the parent group weights into a matrix.

Definition 10.3.3 (*Parent group weights matrix*)**.** If an agent has m parent agent groups $\{G_1, ..., G_m\}$, with n atomic task types $\{ap_1, ..., ap_n\}$, and r types of resource to manage $\{rp_1, ..., rp_r\}$, we define the *parent group weights matrix* of an agent g as:

$$\mathbf{PW} = \begin{bmatrix} \mathbf{PW}_{rp_1} \\ \mathbf{PW}_{rp_2} \\ \vdots \\ \mathbf{PW}_{rp_r} \end{bmatrix} \tag{10.6}$$

where, $\forall \mathbf{PW}_{rp} \in \{\mathbf{PW}_{rp_1}, \mathbf{PW}_{rp_2}, ..., \mathbf{PW}_{rp_r}\}$:

$$\mathbf{PW}_{rp} = \begin{bmatrix} weight(g, G_1, ap_1, rp) & \dots & weight(g, G_1, ap_n, rp) \\ weight(g, G_2, ap_1, rp) & \dots & weight(g, G_2, ap_n, rp) \\ \vdots & \dots & \vdots \\ weight(g, G_m, ap_1, rp) & \dots & weight(g, G_m, ap_n, rp) \end{bmatrix} \tag{10.7}$$

In future definitions, we use the shorthand $\mathbf{PW}_{rp}$ for the row matrix in $\mathbf{PW}$ corresponding to the resource type rp. We additionally define a helper function to index into $\mathbf{PW}$.

Parent group weights index Given a parent agent g, a task type ap, and a resource type rp, we define an index into the matrix $\mathbf{PW}$ as $index\colon \mathcal{G} \times \mathcal{AP} \times \mathcal{RP} \rightarrow \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0$.

10.3.2 Combining resource allocation models

Although a child agent models resource allocations for each of its parent agent group individually, we assume its real resource allocation cannot be instantly changed (e.g. a car in a V2X system may allocate some battery power to its wireless transmitter, however, to reallocate this resource to another component such as a LIDAR system, would involve a delay, and is not instantaneous). Due to this reallocation delay, the child agent must choose a single model, an aggregate of all of its parent agent groups' models, to allocate its available resources. To do this, we use the frequency of incoming atomic task types, and the entropy of the differing models, to weight the contribution of each of the models to the single model.

Incoming task frequency The reasoning for using task frequency is as follows. In a situation where we have two sets of parent agents groups G_1 and G_2 allocating tasks of type ap to a child agent g, we will have corresponding sets of resource weights W_1 and W_2 respectively for a resource res. If the frequency of incoming tasks from group G_1 is significantly greater than that of G_2, we want the child agent's actual resource allocation to be closer to the values in W_1 as we will generate more system utility through applying that allocation given the overall distribution of incoming tasks.

Parent agent sample count To have the preferred resource allocation of some parent agent groups contribute to the child agent's aggregate resource allocation more strongly than that of others, we utilise sample counting to measure the relative frequency of all tasks received from each parent agent group. So we define the *parent agent allocation count* as a count of the number of times any task has been allocated to an agent g from a member of a given set of agents G, $allocnum\colon \mathcal{G} \times 2^{\mathcal{G}} \rightarrow \mathbb{N}_0$.

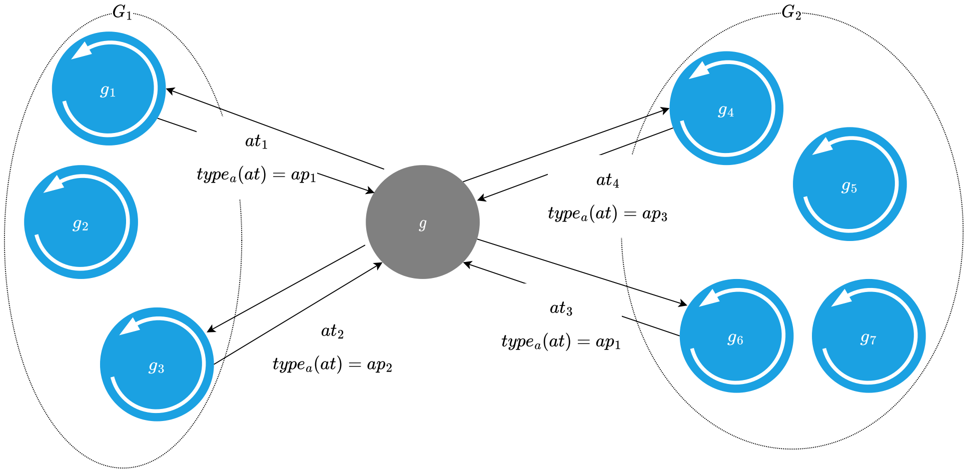


Figure 10.1: Resource allocation using parent groups. *An agent g is allocated atomic tasks from two parent groups G_1 and G_2. Agents in G_1 allocate the agent tasks of type ap_1 and ap_2, requiring the use of an agent's resource res_a. Agents in G_2 allocate g tasks of type ap_1 and ap_3, requiring resource res_b, where $type_r(res_a) \neq type_r(res_b)$. Assuming all tasks are of equal frequency and value to the parent agents, g will weight resources of type $type_r(res_a)$ uniformly between task types ap_1 and ap_2 for parent group G_1, and fully weight the resource to ap_1 for G_2. There will be a full weighting of $type_r(res_b)$ for G_2 with task type ap_3 as its the only parent group allocating a task type that requires it. However, there will also be a uniform weighting of $type_r(res_b)$ for G_1 as weighting per parent group for each resource must sum to 1 (See Definition 10.3.2). Since this resource is never required by this group, the weighting is ignored by g when actually allocating its resources. How this is done is covered in Section 10.3.2 on resource weight blending.*

The entropy of resource weightings

We justify using the relative entropy of resource weighting models as a factor in aggregating them together from the observation that low relative entropy[2] across a parent group can indicate one or more of the following situations;

- the overall outcome of the combined tasks in that group are not strongly influenced by the resource allocation policy of the child agent;
-
- the distribution of incoming tasks for that model is less predictable than the others, possibly due to the changing atomic task allocations of parent agents, or those parent agents receiving varying composite tasks.

Parent group weights entropy

We therefore use the principle of maximum entropy of resource allocation[323] to increase the influence of the resource weightings of parent agent groups where;

- the value of composite tasks for agents in that group are strongly impacted by the child agent's resource allocation policy;
- where the atomic tasks the parent agents are allocating to the child agent do not result in atomic task values that are very similar;
- the group provides stable distributions of atomic task allocations to the child agent[324], [325].

Definition 10.3.4 (*Parent group weights entropy*)**.** Given an agent g with a parent group G, that can receive atomic task types AP, then the *parent group weights entropy* for a resource type rp, is the relative entropy[3] of the parent group's resource weights, given by $kullback\colon \mathcal{G} \times 2^{\mathcal{G}} \times 2^{\mathcal{AP}} \times \mathcal{RP} \to \mathbb{R}$, where:

$$kullback(g, G, AP, rp) = \sum_{ap \in AP} weight(g, G, ap, rp).\log\left(\frac{weight(g, G, ap, rp)}{1/|AP|}\right) \tag{10.8}$$

Aggregating models with a blending function

To aggregate the learnt models of resource allocation for each parent agent group into one model we define the *resource weights blending function*, a function that balances the relative sample counts and entropy of each parent group as defined previously. We use *sum-normalisation*[4] to balance the impacts of entropy and relative sample frequency on the final resource weights.

Definition 10.3.5 (*Resource weights blending function*)**.** Given an agent g with parent agent groups, $group(g, G) = \{G_1, \ldots, G_m\}$, weighting the allocation of a resource of type rp, amongst a set of atomic task types AP, the *resource weights blending function* is defined by:

$$\begin{aligned} blend(g, G, AP, rp) = {} & sumnorm(\begin{bmatrix} kullback(g, G_1, AP, rp) & \ldots & kullback(g, G_m, AP, rp) \end{bmatrix}^T) \\ & + sumnorm(\begin{bmatrix} allocnum(g, G_1) & \ldots & allocnum(g, G_m) \end{bmatrix}^T \end{aligned}$$

We use the shorthand BL for the vector generated by this function.

[2] As compared to a uniform distribution of resources amongst task types.

[3] The relative entropy or Kullback-Liebler divergence of distribution P from the uniform distribution, U is defined as $\sum P(x) \log \frac{P(x)}{U(x)}$

[4] Sum-normalisation as defined by the equation $sumnorm(X) = \{\frac{x_i}{\sum X}\}_{\forall x_i \in X}$

Combining resource weights

With the grouping of parent agents' respective resource weights, and a blending matrix based on relative parent-group task frequency and per-group entropy of these resource weights, we can now define a function that will combine the two to give a single resource allocation model for each child agent.

Definition 10.3.6 (*Combined resource weights function*). The *combined resource weights function* takes a blending vector BL, a resource weights matrix $\mathbf{PW}$, and a resource type rp and outputs a vector of resource weights for the given resource for atomic task types $\{ap_1, \ldots, ap_n\}$[5]:

$$combined(BL, \mathbf{PW}, rp) = softmax(sumnorm(BL^T)\mathbf{PW}_{rp}) \tag{10.9}$$

These values are used by the child agent as its resource allocation model to apply to real resources. *Softmax normalisation*[6] is used to ensure the final resource weights sum to 1 as required.

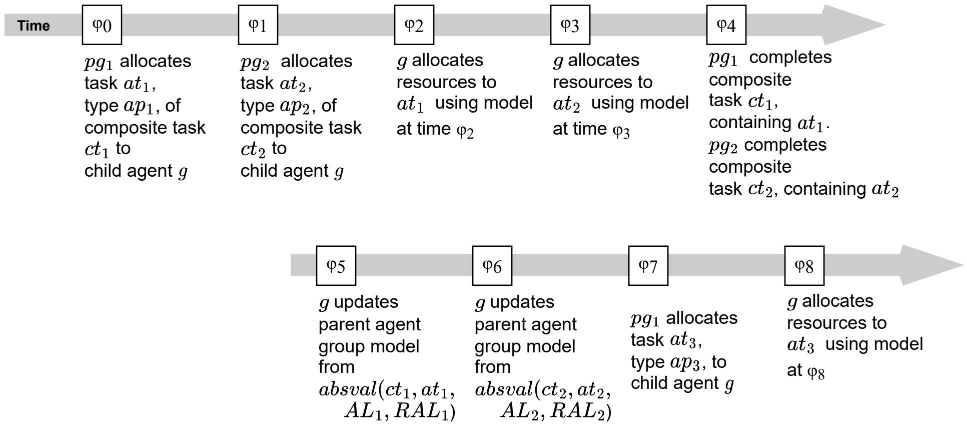
Example 10.3.2 (*Model aggregation in a V2X system*). An agent g controlling a vehicle is receiving tasks from other vehicles combined into three separate parent groups. Vehicles in parent group G_1 frequently request position and speed updates as they are near to g, giving a high sample count. Those in G_2 are not nearby, but are entering a congested area so congestion task results are more valuable to them than others, so their resource weights have a high relative entropy. Finally, vehicles in parent group G_3 are neither nearby nor nearing congestion so tasks are of relatively equal value to them. In these circumstances, where $allocnum(g, G_1) \gg allocnum(g, G_2), allocnum(g, G_3)$, the combined resource weight function will output an aggregated model that more closely resembles the individual model for G_1, maximising the absolute values returned from highly frequent tasks. Similarly, as $kullback(g, G_2, AP, rp) \gg kullback(g, G_1, AP, rp), kullback(g, G_3, AP, rp)$, the combined resource weight function will be closer to that of G_2, to maximise the value returned due to tasks that are more important to their respective parent agents. As the resource model from G_3 is neither frequent, nor generates significantly more value if a particular resource model is applied, it has low sample count and low relative entropy and so has its impact on the final model reduced in both cases.

Past resource allocation effects

Over a period of time a child agent's resource allocation model will change as it adapts to the absolute task values returned by multiple parent agents, with each change in value for one atomic task type necessarily changing the values for the other atomic task types in the same group. This means the weight of an atomic task may have had contributions to it arising from other, previous atomic task values used to alter other tasks (see Figure 10.2). This challenge in attributing value to actions when multiple past actions may have contributed to an outcome is an example of the *credit assignment problem*[326] commonly found in reinforcement learning systems.

[5] Note that the blending matrix BL has dimension $1 \times |group(g, G)|$, and the resource weights matrix $\mathbf{PW}$ for a given resource type rp, a dimension of $|group(g, G)| \times |AP|$, resulting in the function output being of dimension $1 \times |AP|$.

[6] Softmax normalisation is defined as $softmax(X) = \{\frac{e^{x_i}}{\sum_{j=1}^{|X|} e^{x_j}}\}_{\forall x_i \in X}$



Time
φ0
pg_1 allocates task at_1, type ap_1, of composite task ct_1 to child agent g
φ1
pg_2 allocates task at_2, type ap_2, of composite task ct_2 to child agent g
φ2
g allocates resources to at_1 using model at time φ2
φ3
g allocates resources to at_2 using model at time φ3
φ4
pg_1 completes composite task ct_1, containing at_1. pg_2 completes composite task ct_2, containing at_2
φ5
g updates parent agent group model from $absval(ct_1, at_1, AL_1, RAL_1)$
φ6
g updates parent agent group model from $absval(ct_2, at_2, AL_2, RAL_2)$
φ7
pg_1 allocates task at_3, type ap_3, to child agent g
φ8
g allocates resources to at_3 using model at φ8

Figure 10.2: Model updates with asynchronous task allocation. *A parent agent pg_1 allocates an atomic task at_1, of type ap_1 to a child agent g, and parent agent pg_2, an atomic task of type ap_2. On completing the corresponding composite task ct, pg returns the component task absolute values $absval(ct_1, at_1, AL_1, RAL_1)$ and $absval(ct_2, at_2, AL_2, RAL_2)$ to g, which updates the weights for its resource allocation model. Next, a new task of type ap_3 is allocated and completed. When $absval(ct_3, at_3, AL_3, RAL_3)$ is returned there is a problem in deciding which, and how much, each previous model update affected the value of ap_3.*

Eligibility trace update To take account of this effect we use a standard technique, a *replacement eligibility trace matrix*[166], to apply the effect of absolute task values backwards through past model changes. This enables the MG-RAO algorithm to attribute some of the current model quality to past resource allocation changes.

Definition 10.3.7 (*eligibility trace matrix*)**.** A child agent's *eligibility trace matrix* **E** has the same shape as the parent group weights matrix **PW** with each element initialised to 0. Given an atomic task type ap, a parent agent pg, and a fixed decay factor γ', updates are carried out to each element $\langle i, j, k\rangle$ of the eligibility trace matrix using the *eligibility trace update* defined below.

$$updatetrace(\mathbf{E}, pg, ap, rp, \gamma') = e_{ijk} \longleftarrow \begin{cases} 1 & \text{if } \langle i, j, k\rangle = index(pg, ap, rp) \\ \gamma' e_{ijk} & \text{if } e_{ijk} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{10.10}$$

This sets the $\langle i, j, k\rangle$ element in the eligibility trace matrix that corresponds to the element $weight(g, G_i, ap_j, rp)$ in the parent group weights matrix **PW** to one. Where all other elements are multiplied by the decay factor γ'. The effect of this is that the eligibility trace matrix measures how recently each task type has been allocated by each parent agent group.

Updating a resource weights matrix

When an absolute value *absval* is received by an agent as a reward from a parent agent for completing a task, the resource weights matrix **PW** is updated using the eligibility trace **E** to define which indices in the resource weights matrix are updated, and by what fraction of the absolute value:

$$updateweights(absval, \mathbf{PW}, \mathbf{E}) = \mathbf{PW} + \alpha^{'}.absval.\mathbf{E} \tag{10.11}$$

Where $\alpha^{'}$ is a constant value that defines the rate of update of the resource weight matrix.

10.3.3 Extending agent state for resource allocation

Initially in Chapter 8, Section 8.3, the state of an agent contained its neighbourhood and knowledge. In Chapter 9, Section 9.4.6, we extended this state to include an agent's action samples, TSQM, and Q-table, in order to use the ATA-RIA algorithm for task allocation. As our final extension to the agent state we include the agent's resource weights and eligibility trace as part of its state.

Definition 10.3.8 (*Agent State (Extended)*)**.** Given an agent $g = \langle id, c, r, \delta_k, \delta_n \rangle$, we define its state as a tuple $\langle K, N, SP, \Lambda, Q, \mathbf{PW}, \mathbf{E} \rangle$, where:

- $K \subseteq G$ is the knowledge of the agent.
- $N \subset K$ is the neighbourhood of the agent.
- SP is the set of action samples of the agent.
- Λ if the TSQM of the agent.
- Q is the Q-table of the agent.
- **PW** is the parent group weights matrix of the agent.
- **E** is the eligibility trace matrix of the agent.

We will use the notation $\mathbf{PW}(g)$, and $\mathbf{E}(g)$ to refer to the resource allocations, resource weights, and eligibility trace, respectively of an agent g.

10.4 The multi-group resource allocation optimisation (MG-RAO) algorithm

With parent groups, resource weights, blending vectors, and the combined resource weights function defined, we can bring these together to form the *multi-group resource allocation optimisation (MG-RAO) algorithm* to optimise for system utility. This algorithm solves the problem of resource allocation in dynamic systems through the use of two sub-algorithms. The MG-RAO (update) algorithm learns the resource weightings that maximise the component task values of parent groups' atomic task allocation distributions. The MG-RAO (weight) algorithm combines these into a resource weightings model that is applied across all incoming tasks. The process followed when a task is assigned to a parent agent is shown in Figure 10.3 with the flow of weight modelling and combination shown in Figure 10.4.

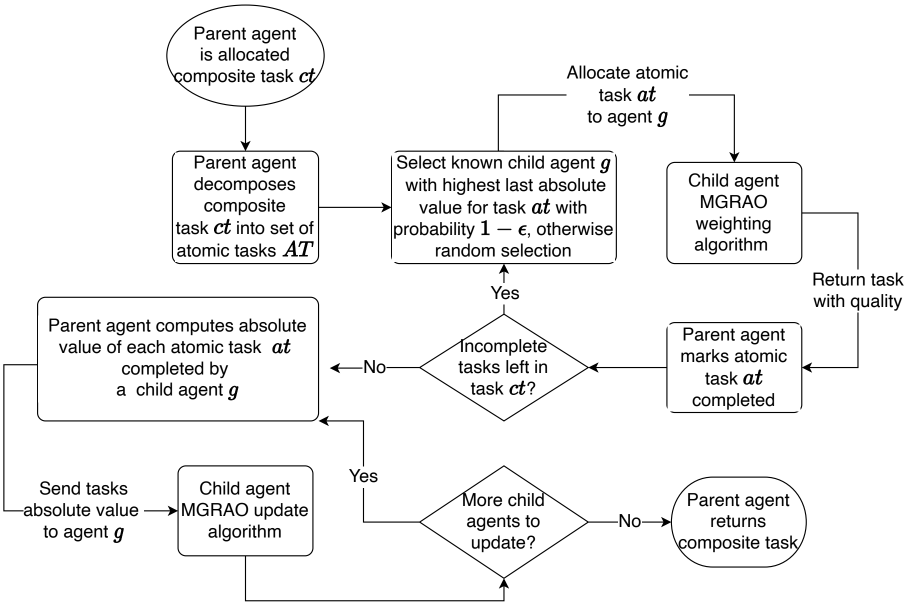

Parent agent is allocated composite task ct
Parent agent decomposes composite task ct into set of atomic tasks AT
Select known child agent g with highest last absolute value for task at with probability $1-\epsilon$, otherwise random selection
Allocate atomic task at to agent g
Child agent MGRAO weighting algorithm
Return task with quality
Parent agent marks atomic task at completed
Incomplete tasks left in task ct?
Yes
No
Parent agent computes absolute value of each atomic task at completed by a child agent g
Send tasks absolute value to agent g
Child agent MGRAO update algorithm
More child agents to update?
Yes
No
Parent agent returns composite task

Figure 10.3: High-level view of the MG-RAO algorithm. *This diagram illustrates how MG-RAO algorithm workflow progresses as atomic tasks are allocated to an agent, and how it uses the algorithm to update its weighting of resource allocations to optimise performance.*

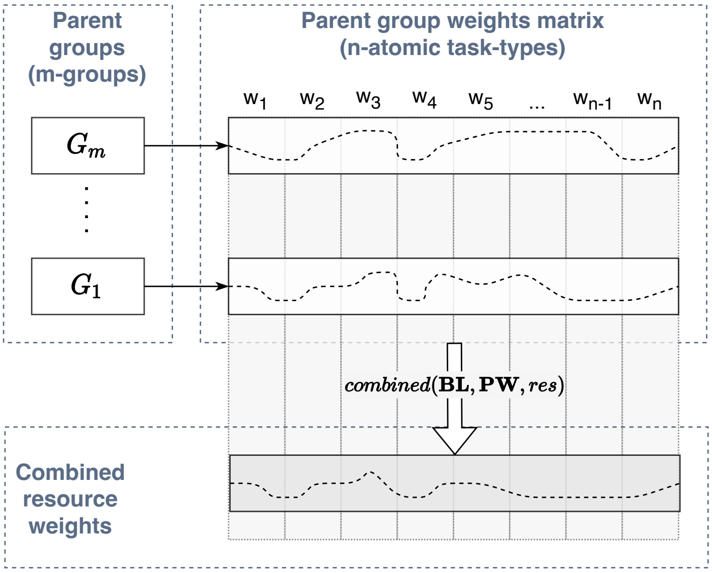

Parent groups (m-groups)
Parent group weights matrix (n-atomic task-types)
w_1 w_2 w_3 w_4 w_5 ... w_{n-1} w_n
G_m
G_1
$combined(\mathbf{BL}, \mathbf{PW}, res)$
Combined resource weights

Figure 10.4: MG-ROA algorithm weight blending. *Weight blending allows agents to model their resource allocation for multiple parent agent groups, and then combine these separate models into a final model that will decide the actual resource allocation enacted.*

The MG-RAO update algorithm

The MG-RAO update algorithm is applied when a child agent cg receives an absolute task value from a parent agent pg, in a parent agent group G, for completing an atomic task type $ap = type_a(at)$ of the parent's composite task ct.

1. We update the eligibility trace matrix using Equation 10.10 [line 2].
2. The updated eligibility trace matrix is then multiplied by the value of $absval$, and a fixed learning rate factor $\alpha^{'} \in [0, 1]$, then added to resource weights matrix of the agent g, for the resource res, $\mathbf{PW}_{rp}(g)$ [line 3].
3. The resulting matrix of weights is then sum-normalised row-wise for each resource type [line 4].

ALGORITHM 5: The multi-group resource allocation optimisation (MG-RAO) update algorithm

Input: at, an atomic task allocated to agent g.
Input: pg, the parent agent that allocated the atomic task at.
Input: res, the resource requiring allocation.
Input: $absval$, the absolute value returned for the completed atomic task.
Input: $\langle \mathbf{PW}, \mathbf{E} \rangle$, the agent state resource weight matrix and eligibility trace for agent g.
Output: $\langle \mathbf{PW}^{'}, \mathbf{E}^{'} \rangle$, updates to the resource weight matrix and eligibility trace for agent g.

```
  // Get the atomic task's type
1 $ap \leftarrow type_a(at)$
  // Update the eligibility trace
2 $\mathbf{E}^{'} \leftarrow updatetrace(\mathbf{E}, pg, ap, rp, \gamma')$
  // Apply the eligibility trace to resource weights
3 $\mathbf{PW}^{''} \leftarrow updateweights(absval, \mathbf{PW}, \mathbf{E})$
  // Sum-normalise resource weight matrix rows per $\mathbf{PW}_{rp}$
4 $\mathbf{PW}^{'} \leftarrow sumnorm(\mathbf{PW}^{''})$
5 return $\langle \boldsymbol{PW}^{'}, \boldsymbol{E}^{'} \rangle$
```

The MG-RAO weighting algorithm

The MG-RAO weighting algorithm is used when an atomic task is being performed by a child agent.

1. We first generate the resource blending function from the set of atomic task types in the system and the type of the resource [line 3].
2. We use the combined resource weight equation to generate the $1 \times |AP|$ matrix of combined resource weights from the resource weights matrix $\mathbf{PW}$ of the agent g [line 4].
3. We get the total resource availability of resource $type_r(res)$ [line 5].
4. For each of the atomic task types in the system, we do the following steps:
   (a) We find the parent-task type index for the specific parent agent pg, atomic task type ap, and resource type rp [line 7].
   (b) We then use the index values to select the correct weight from the combined resource weights, then multiply this value by the total available resource to the child agent [line 8].
   (c) The resource allocation is updated with the new weighting for the agent g and atomic task type ap [lines 9 - 10].

ALGORITHM 6: The multi-group resource allocation optimisation (MG-RAO) weighting algorithm

```
Input: $g$, an agent that is performing the task $at$.
Input: $pg$, the parent agent that allocated the atomic task $at$.
Input: $res$, the resource requiring allocation.
Input: $\langle \mathbf{PW}, \mathbf{E} \rangle$, the agent state resource weight matrix, and eligibility trace for agent $g$.
Result: updates to the resource allocation of the system, $RAL$.

$ap \leftarrow type_a(at)$
$rp \leftarrow type_r(res)$
// Calculate the resource weights blending vector
$BL \leftarrow blend(g, G, AP, rp)$
// Calculate the combined resource weights of the resource $res$
$\mathbf{C} \leftarrow combined(BL, \mathbf{PW}, rp)$
// Get the total resource available for the agent's resource type $rp$
$totalres = resavailable(g, rp, RAV);$
foreach $ap \in AP$ do
    // Find the index of the column for $rp$ in the resource matrix
    $\langle i, j, k \rangle \leftarrow index(pg, ap, rp)$
    // Multiply the agent's resource availability for $res$ by its weighting for $ap$
    $w \leftarrow totalres \times \mathbf{C}_{ij}$
    // Remove previous weight for $ap$ and $rp$ from system state resource allocation
    $RAL \leftarrow RAL \setminus \{\langle g, ap, w' \rangle : \langle g, ap, w' \rangle \in RAL, \exists w' \in \mathbb{R}\}$
    // Update the system state resource allocation with the new weight
    $RAL \leftarrow RAL \cup \{\langle g, ap, w \rangle\}$
end
```

10.5 Evaluation

We simulated four systems to evaluate the MG-RAO algorithm. A *single-child* system was used to evaluate the performance of MG-RAO where a child agent's incoming tasks had a stable distribution, as the parent agents competed over the resource allocation of one agent only. In the *multi-child* system parent agents could choose between a number of child agents to allocate atomic tasks to, with a random probability, ϵ, that they would allocate to a non-optimal child agent. Non-optimal child agents were selected based on fixed-temperature *Boltzmann selection*[7] of their absolute task values, as found by the parent agent's previous task allocations. This system tested the MG-RAO algorithm where child agents' parent groups had variable incoming task distributions. To evaluate the effect of dynamism on performance we simulated a *volatile* system where parent agents had a fixed probability of leaving or re-joining the system each episode. Finally, the *large system* tested the effect of varying parent group size when there were many parent agents allocating tasks to each child agent, to examine the scalability of the algorithm.

Labels for the algorithms and configurations used in the simulations are described in Table 10.1. General system parameters and individual system parameters are shown in Tables D.1 and D.2 respectively, included in Appendix D. The composite task frequency distribution introduced the same fixed set of tasks over a defined period, giv-

[7] The probability of choosing a child agent cg with atomic task quality q_{cg} from N other agents is, $P(a_i) = \dfrac{e^{(p_i/\tau)}}{\sum_{j=1}^{N} e^{(p_j/\tau)}} cg$

Algorithm	Summary
<uniform>	Resource allocation weights of child agents are fixed to uniform values.
<mgrao-1:1>	This system uses only one parent agent group per-child agent rather than multiple groups. This is similar to state-of-the-art distributed Q-learning algorithms.
<mgrao-max>	Every parent agent is placed in its own parent-agent group.
<mgrao-x:y>	These algorithms have an x:y ratio of parent-agent groups to each child-agent.

Table 10.1: Summary of algorithm labels. *Label and descriptions of the uniform, mgrao-1:1, mgrao-max, and mgrao-x:y algorithms*

ing each *episode* of the system. The parent agent groups for each child agent were fixed throughout the simulation. Each child agent's available resources were set at system start time in the range $(0, 1] \in \mathbb{R}_{>=0}$ drawn randomly from a *normal distribution*[8].

Algorithm	% from <uniform>	% from <mgrao-max>	% of <mgrao-max>
<mgrao-max>	28.0%	−0.0%	100.0%
<mgrao-1:1>	23.4%	−4.6%	83.7%

Table 10.2: Experimental results for single child system after 100 episodes.

Algorithm	% from <uniform>	% from <mgrao-max>	% of <mgrao-max>
<mgrao-max>	23.8%	−0.0%	100.0%
<mgrao-1:1>	21.4%	−2.4%	90.0%

Table 10.3: Experimental results for multi-child system after 100 episodes.

Results for the single-child, multi-child, volatile and large systems are shown in Tables 10.2, 10.3, 10.4 and 10.5. The percentage system utility improvement for the system using the algorithm specified rather than uniform resource allocation is shown. Percentage utilities are also included for each algorithm in comparison to the <mgrao-max> algorithm for each system. The p-values showing the statistical significance of the system utility values are shown in Appendix D Table D.3.

Single child agent performance

As seen in Figure 10.5, <mgrao-max> shows a 28.0% improvement in performance as compared to the <uniform> case. It also performs 4.6% better than <mgrao-1:1> after 100 episodes. Although the <mgrao-1:1> algorithm initially learns to improve its allocation policy more quickly than <mgrao-max>, by episode 10 its performance flattens out and <mgrao-max> surpasses it. As well as the performance improvements in allocation optimality, these results demonstrate how the learning of multiple resource

[8] A normal distribution defined by values in $X \sim \mathcal{N}(\mu, \sigma^2), \mu = 0.5, \sigma = 0.2$

Algorithm	% from <uniform>	% from <mgrao-max>	% of <mgrao-max>
<mgrao-max>	7.1%	−0.0%	100.0%
<mgrao-1:1>	3.3%	−3.8%	46.5%

Table 10.4: Experimental results for volatile system after 100 episodes.

Algorithm	% from <uniform>	% from <mgrao-max>	% of <mgrao-max>
<mgrao-1:1>	14.7%	−4.9%	75.1%
<mgrao-2:1>	17.2%	−2.4%	87.7%
<mgrao-5:1>	17.9%	−1.8%	91.0%
<mgrao-10:1>	18.9%	−0.6%	96.6%
<mgrao-25:1>	19.5%	−0.1%	99.4%
<mgrao-max>	19.6%	−0.0%	100.0%

Table 10.5: Experimental results for large system after 100 episodes.

allocations for groups of agents before blending allows the algorithm to learn a more complex function approximation.

Performance with choice of multiple child agents From Figure 10.6 we can see that <mgrao-max> performs 23.8% better than <uniform> and 2.4% better than <mgrao-1:1> after 100 episodes. As the <mgrao-1:1> simulation treats all of a child agent's incoming atomic tasks as one, it does not retain parent agent task allocation patterns through time as well as when multiple parent agent groups are used. Therefore, when a parent agent stops allocating tasks to it, knowledge of the learnt task allocation distribution for that agent is quickly lost. With <mgrao-max>, different optimal resource weighting distributions are learned for each parent-agent group. When a parent agent's task allocations are intermittent, the child agent's knowledge of that agents best-known resource weightings distribution persists for longer. Thus, when a parent agent's task allocation to a child agent is temporarily disrupted through effects such as exploring allocating atomic tasks to other agents, or losing connectivity, the child agent is able to reuse past learnt information when the parent agent start allocating tasks to it again.

Performance under agents leaving and joining the system In the volatile system simulation, shown in Figure 10.8, we see <mgrao-max> perform 7.1% better than <uniform>. In contrast, <mgrao-1:1> only achieves 46.5% of the improvement of <mgrao-max> under the same conditions. As parent agents leave and join the system at random, the use of parent-agent groups allows MG-RAO to retain the knowledge the child agent has about these agents' task allocation patterns for a longer period of time than when using fewer parent-agent groups. As agents that had previously left the system rejoin (or regain connectivity), the child agent can re-apply this previous knowledge without having to entirely re-learn the parent agent's task allocation distribution. This means <mgrao-max> will have a higher system utility under volatile conditions than <mgrao-1:1>, which loses this knowledge more quickly.

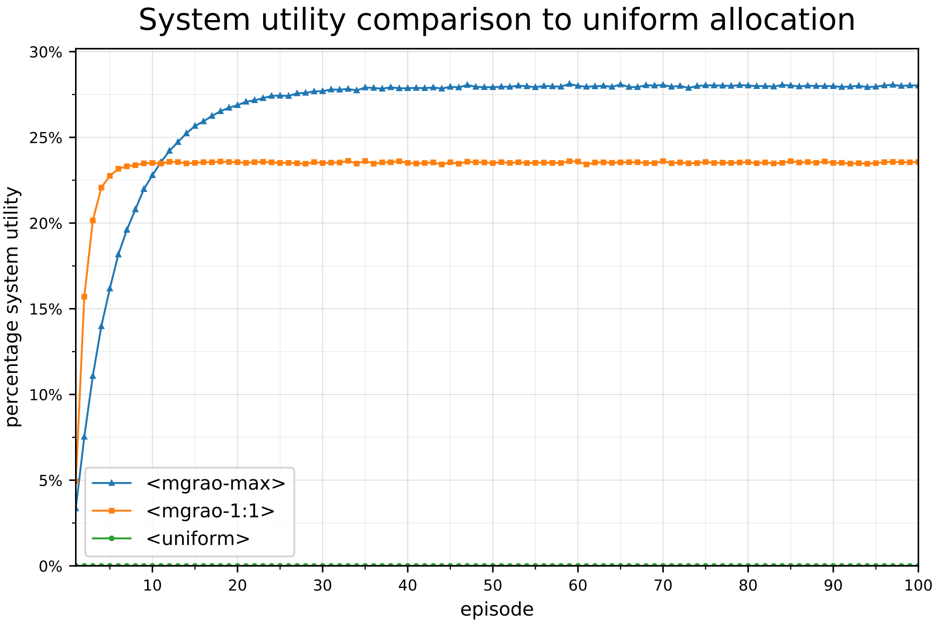


Figure 10.5: *System utility comparison to uniform allocation - single child agent*

Performance changes with parent-agent group size in a large system

In the large system <mgrao-max> uses the maximum number of parent-agent groups and achieves the best performance, at 19.6% over <uniform>. The algorithms <mgrao-25:1>, <mgrao-10:1>, <mgrao-5:1>, <mgrao-2:1>, <mgrao-1:1> achieve 99.4%, 96.6%, 91.0%, 87.7%, and 75.1% of the performance of <mgrao-max> respectively. Though <mgrao-1:1> was still 14.7% above <uniform> performance. These results demonstrate how the performance of the more scalable, size-constrained parent agent group algorithms still perform close to the algorithm's best evaluated performance, given by <mgrao-max>, where unlimited resources dedicated to learning are assumed. In the <mgrao-x:y> configurations, MG-RAO optimises its resource usage based on the distribution of task allocations and the per-parent agent component task values that result from them, rather than requiring individually learnt values for each agent. As such, modelling based on aggregations of these values can perform well, depending on factors such as their similarity, or how stable the values are for individual parent agents.

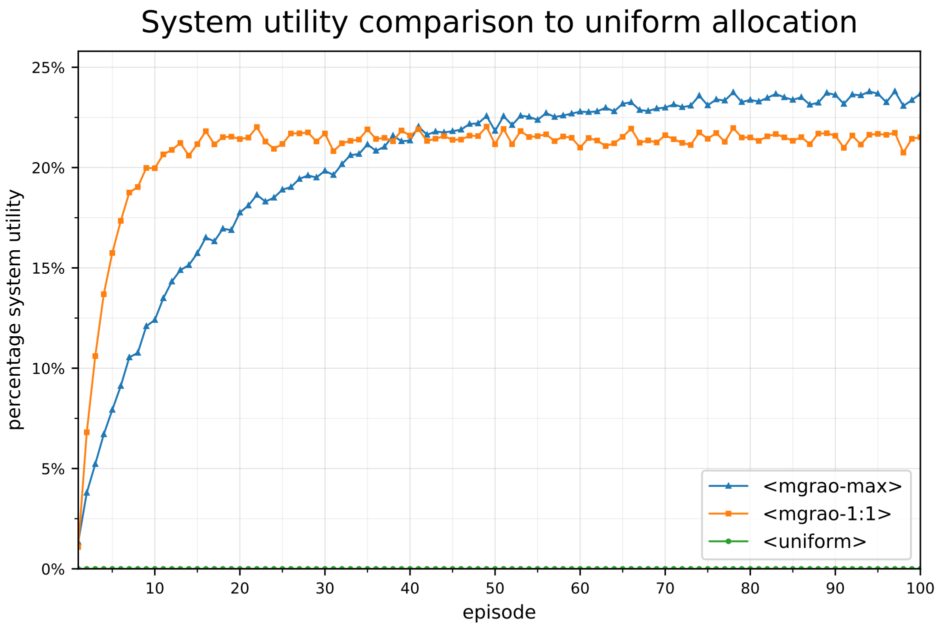


Figure 10.6: *System utility comparison to uniform allocation - multiple child agents*

Overall performance

Overall, the MG-RAO algorithm shows a $23-28\%$ improvement over fixed resource allocation in the simulated environments. Results also show that, in a volatile system, using the MG-RAO algorithm configured so that child agents model resource allocation for all agents as a whole has 46.5% of the performance of when it is set to model multiple groups of agents. These results demonstrate the ability of the algorithm to perform well in optimising for resource allocation problems in dynamic multi-agent systems.

As a child agent learned models of the value to the parent agents of the tasks assigned to it, it was able to better distribute its resources to maximise utility. The use of multiple models of incoming tasks split across parent groups allowed a more detailed resource allocation model to be learned than when only one group was used, and also gave the algorithm robustness under volatility and variations in the distribution of tasks.

Evaluation was performed in systems ranging from $1-3$ child agents being allocated tasks from between $10-50$ parent agents, representing common ranges found in real-world multi-agent problems such as those found in V2X[327] and warehouse automation systems[328]. The addition of parent groups, and their scalable performance, is important to the algorithm's applicability across a large range of system sizes. As they aggregate parent agents into a fixed number of groups, they act as a constraint on the resource overhead required to run the algorithm. This helps the solution be applicable to larger systems.

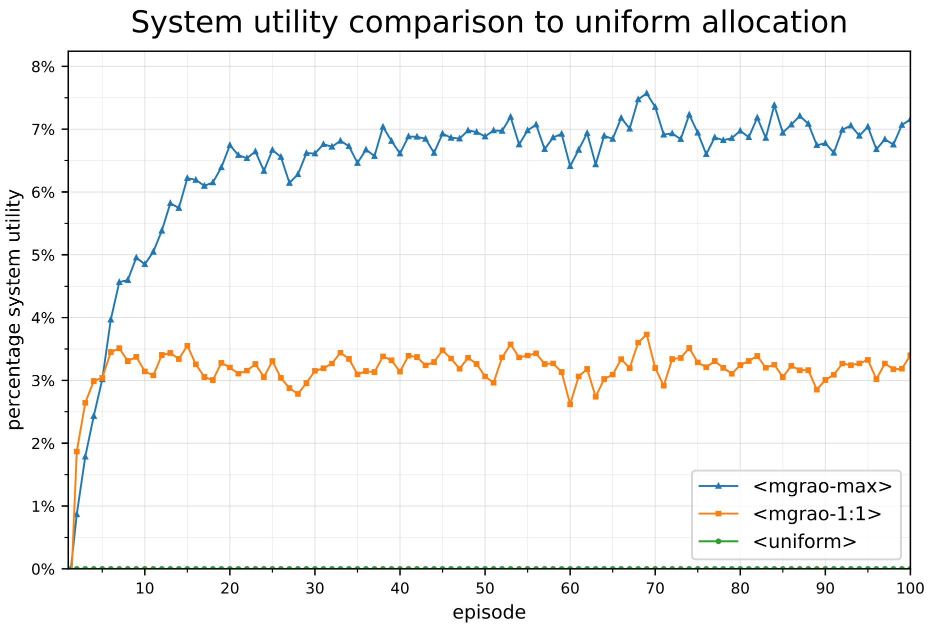


Figure 10.7: *System utility comparison to uniform allocation - volatile system*

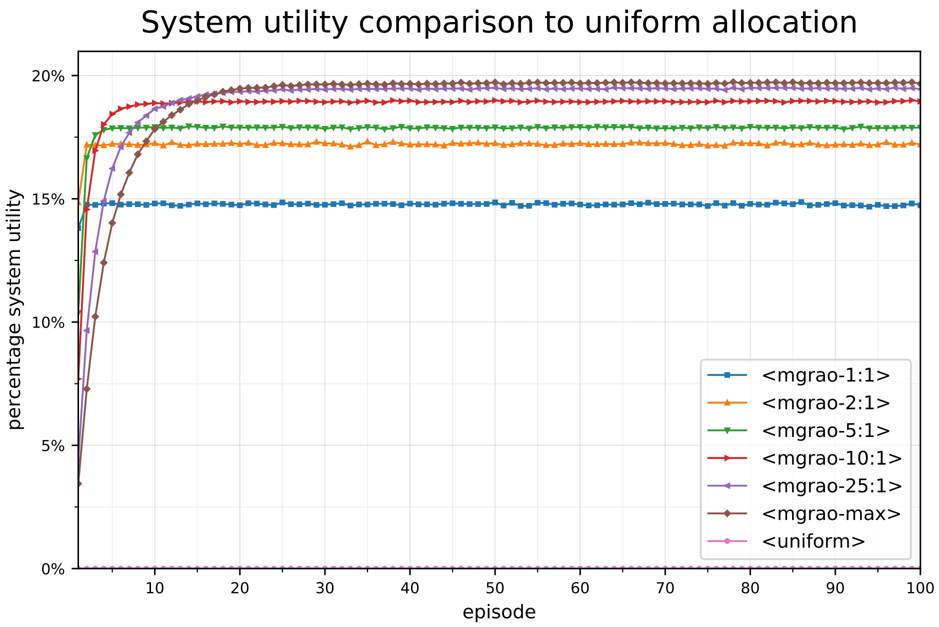


Figure 10.8: *System utility comparison to uniform allocation - large system*

10.6 Summary

The work in this chapter looked at the problem of how agents can distribute their resources across multiple tasks, which have been allocated to them as part of other agents' composite tasks, to improve the utility of the system. In the next chapter we will combine the work from Chapter 9 alongside that of this chapter, to form a combined solution to task and resource allocation.

Chapter 11

Hierarchical task allocation

In our previous algorithms on task and resource allocations, agents coordinated with each other directly. The following chapter extends this coordination so that, in order to find better quality task allocations, agents can learn to allocate tasks indirectly, to agents beyond their immediate, known neighbours. In doing so, we develop self-organising structures based around completing tasks, where agents can assume a variety of possible roles (Contribution 3).

11.1 Introduction

In Chapter 9 we introduced the task allocation problem and our algorithms to tackle it. Chapter 10 extended the system, and we developed our solution to the optimisation of the allocation of resources by agents when completing tasks. In this chapter we integrate these two solutions and enhance it further to allow agents to reallocate atomic tasks to other agents[1]. This means there can now be a chain of atomic task allocations of atomic tasks in the system before they are finally executed.

We progress our work on our hierarchical task allocation solution as follows[2]; by defining roles and task paths for atomic tasks; then we develop a unified algorithm (HTAO) where the rewards for task completion are shared amongst agents that participated in a task path for the completion of that atomic task. This includes the integration of our previous algorithms on task allocation (ATA-RIA) and resource allocation (MG-RAO).

Chapter structure

Section 11.2 covers the roles, task paths, and specialisms. In Subsection 11.2.1 we define the roles agents can assume in completing an atomic task. We develop task paths in Subsection 11.2.2 to take account of these roles. In Subsection 11.2.3 we explore the strategies that agents may develop when taking part in task paths and in interacting with other agents in general. For clarity, we restate the final system

[1] i.e. where we have been using the restricted SINGLEALLOC action to allocate tasks we now use the full ALLOC action, see Chapter 8, Section 8.4.

[2] Note, we use the term 'hierarchical' in the context of the self-organisational structure of agents (see Chapter 6, Section 6.5) resulting from task allocation, rather than its use in reinforcement learning literature. Each agent that receives a task from a parent, has responsibility for its allocation selection to a child agent, the return of those results to its parent, as well as the passing of the reward from its parent to its children, forming the hierarchical structure for that task.

definition, agent state, and system state notation in Subsection 11.2.4. Section 11.3 introduces the HTAO algorithm specification, followed by a short explanation of the behaviour of the algorithm in Section 11.4. The performance of the HTAO algorithm is evaluated as part of Chapter 12.

11.2 Coordinating agents to complete atomic tasks

In this chapter, we enable child agents to reallocate atomic tasks to other agents, which means that parent agents can explore the system for better agents to allocate tasks to, beyond those in their neighbourhoods that they know about directly. The cost of these chained allocations is the use of resources by the intermediary relay agents (e.g. using energy for extra transmissions between agents in a WSN) that are reallocating atomic tasks and returning results.

11.2.1 Roles in a task allocation

Roles For each atomic task, agents involved in its execution can have different *roles*, behaviours they take on to help complete the task. The roles agents can assume are dynamic, dependent on the actions chosen by each agent. Agents also assume a single role for each atomic task completion they participate in, but may be part of many different atomic task completions, and so may play multiple, different roles. The aggregation of these roles they choose to play in the system will define an agent's specialism.

Sinks, relays, and executors Chapter 9 introduced the role of parent agent, agents who allocated atomic tasks to other agents for completion, and child agents, who were allocated tasks, then completed those tasks, returning the results to parent agents. Parent agents learned which other agents to allocate atomic tasks to achieve the best performance, and child agents learned how to best allocate their resources to complete those tasks.

As we now extend atomic task allocation to include reallocation of tasks, we need to be more granular in our definition of agent roles in the system. We distinguish between three roles assumed by participating agents in an atomic task execution;

- A *sink* is an agent that receives composite tasks from outside the system[3]. This agent is the sink agent for all atomic tasks that comprise the composite tasks it receives $at \in ct$, $sink\colon \mathcal{AT} \rightarrow \mathcal{G}$;
- A *relay* is an agent that is allocated an atomic task, but does not complete it, instead it reallocates it to other agents. Given an atomic task at, the ordered sequence of agents who act as relays is given by $relays\colon AT \rightarrow seq\ \mathcal{G}$, where the final agent in the sequence is that which allocates the task to the agent that completes the task;
- An *executor* is the agent that completes the atomic task. Given an atomic task at, this agent is given by the mapping $executor\colon \mathcal{AT} \rightarrow \mathcal{G}$.

[3] Note, we use the term 'sink' remain consistent with the standard WSN definitions of nodes such as the base stations[9].

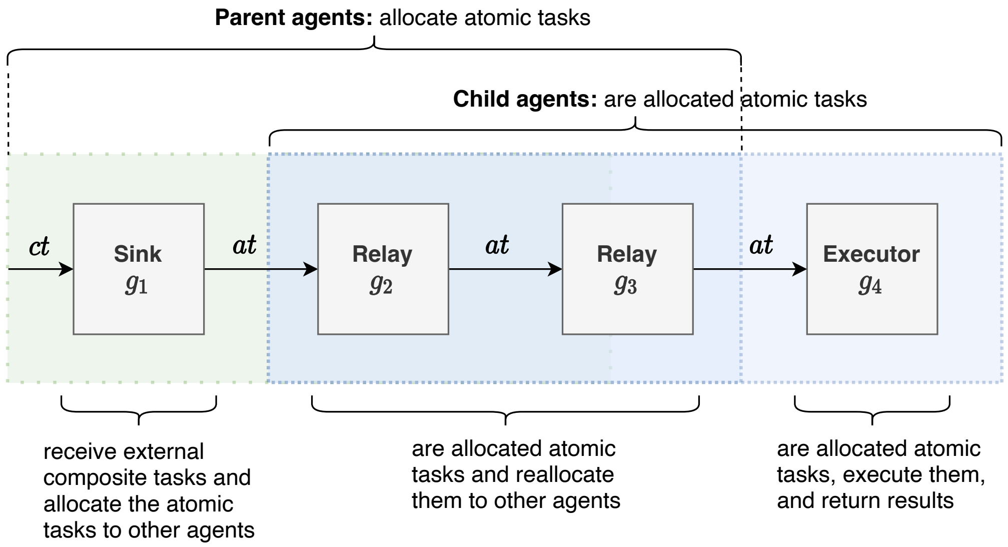

Parent agents: allocate atomic tasks
Child agents: are allocated atomic tasks
ct
Sink
g_1
at
Relay
g_2
at
Relay
g_3
at
Executor
g_4
receive external composite tasks and allocate the atomic tasks to other agents
are allocated atomic tasks and reallocate them to other agents
are allocated atomic tasks, execute them, and return results

Figure 11.1: Types of agent role. *Agents can play multiple roles in the allocation of atomic tasks and their execution. The illustration shows* g_1 *receiving a composite task* ct *from outside the system, it is the sink agent for all the atomic tasks that comprise* ct*:* $\forall at \in ct, sink(at) = g_1$. g_1 *allocates one of the atomic tasks* $at \in ct$ *to the agent* g_2*. In this step* g_1 *acts as a parent agent to* g_2 *for the task* at*, and* g_2 *as the child agent.* g_2 *then allocates* at *to an agent* g_3*, acting as the parent agent, and* g_3 *as the child agent in this step.* g_3 *then allocates* at *to* g_4*, so is the parent and* g_4 *the child.* g_4 *executes the atomic task. Agents* g_2 *and* g_3 *are relays for the atomic task* at*:* $relays(at) = \langle g_2, g_3 \rangle$*, and* g_4 *the executor:* $executor(at) = g_4$*.*

Relationship to parent and child agents

Note that all parent agents are still solving the task allocation problem, but only parent agents that receive component tasks from outside the system are sink agents as well. Similarly, agents who have atomic tasks allocated to them are child agents, but only those who actually complete the tasks are executor agents[4], and therefore must tackle the resource allocation problem of Chapter 10. Relay agents are both child agents in the sense they are allocated atomic tasks to complete (which they then reallocate) and parent agents, as they allocate tasks to other agents. Figure 11.1 shows the different roles and how they overlap in the course of completing composite and atomic tasks.

11.2.2 Task paths

The sequencing of agents assuming a role in a given atomic task execution defines the *task-path*, a mapping of an atomic task to the ordered sequence of agents that work together to complete that task, $path\colon \mathcal{AT} \rightarrow seq\ \mathcal{G}$. In completing an atomic task, the first agent in the sequence is the sink that received the associated composite task,

[4] Note, with no reallocation of atomic tasks possible, all parent agents were sink agents in Chapters 9 and 10, and all child agents were executor agents.

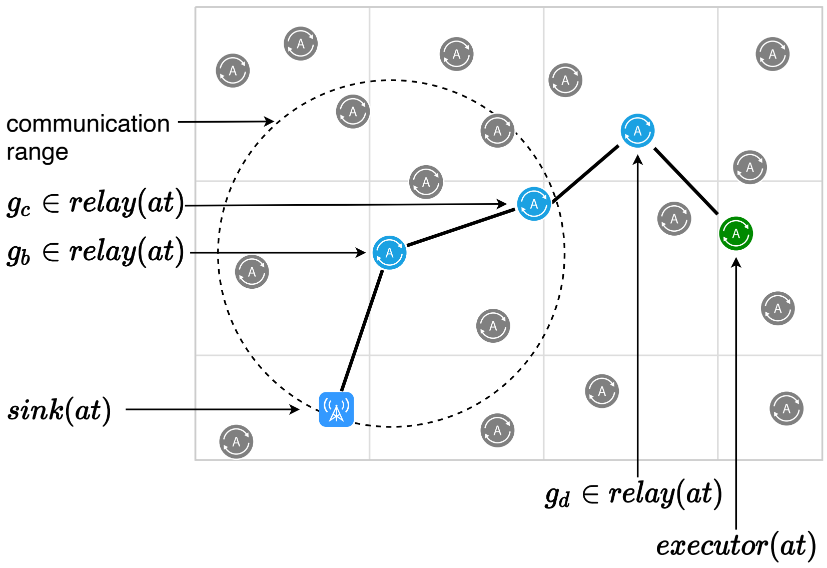


Figure 11.2: Task paths in a WSN. *A sink receives a composite task from an external agent, decomposes it into the single atomic task at, and allocates this to the agent g_b, so $sink(at) = g_a$. g_b relays the task to g_c, an agent in its current transmission range and its neighbourhood, then finally to g_d. g_d completes the task therefore $executor(at) = g_d$ and $relays(at) = \langle g_b, g_c \rangle$. The results of the tasks are then passed from g_d to g_c, then g_b before arriving back at g_a for aggregation and return of the composite task results to the external agent that first made the request. This task path of at is $path(at) = \langle g_a, g_b, g_c, g_d \rangle$.*

the last is the executor that completes the atomic task[5]. (See Figure 11.2):

$$path(at) = \langle sink(at) \rangle, relays(at), \langle executor(at) \rangle \tag{11.1}$$

Task path cost There is a cost to an agent in assuming a role in a task path, the resources that must be used by agents to participate in the process. For example, a sink must communicate with other agents, decompose composite tasks, and assemble the results. Relays must communicate with both sinks, other relays, and executors for the given atomic task. An executor will utilise some of their resources to enable the task to be performed (e.g. activating sensors), and communicate the results.

Definition 11.2.1 (*Task path cost*). Given a task path containing a sequence of agents G, for completing an atomic task at, the *task path cost* for the resource of type rp is given by, $pathcost \colon \mathcal{AT} \times 2^{\mathcal{G}} \times \mathcal{RP} \to \mathbb{R}$.

Example 11.2.1 (*Example of a task path execution flow*). We can describe an example of one possible flow of a composite task execution in a system of agents G as follows;

1. An external agent e sends a composite task ct to an agent g_a, which has a neighbourhood $N(g) = \{g_a, g_b, g_c\}$;

[5] We use tuple notation for lists of sequences[329]. I.e., the concatenation of three sequences is represented as: $\langle a_1, a_2, \ldots \rangle, \langle b_1, b_2, \ldots \rangle, \langle c_1, c_2, \ldots \rangle$

2. g_a decomposes the composite task into atomic tasks $\{at_1, at_2, at_3\}$;
3. g_a can allocate atomic tasks to agents in its neighbourhood, including itself, so;
   (a) g_a allocates at_1 to the agent g_b which completes the task and returns the results to g_a:

   $$sink(at_1) = g_a, relays(at_1) = \emptyset, executor(at_1) = g_b$$

   (b) g_a allocates at_2 to an agent g_c, which then reallocates it to another agent g_d that is not in the neighbourhood of the sink agent g_a. g_d then reallocates to a further agent g_e. The agent g_e completes the task, and returns the results to g_d then g_c, which in turn, returns them to g_a:

   $$sink(at_2) = g_a, relays(at_2) = \langle g_c, g_d\rangle, executor(at_2) = g_e$$

   (c) g_a allocates at_3 to itself and completes it:

   $$sink(at_3) = g_a, relays(at_3) = \emptyset, executor(at_3) = g_a$$

4. g_a aggregates the results of the atomic tasks in ct and returns them to the external agent e. The task paths of the atomic tasks are therefore:

   $$path(at_1) = \langle g_a, g_b\rangle, path(at_2) = \langle g_a, g_c, g_d, g_e\rangle, path(at_3) = \langle g_a, g_a\rangle$$

   .

11.2.3 Developing strategies

As sink, relay, and executor roles are assumed by agents per-atomic task, and agents may participate in multiple task paths, agents may form strategies that take account of these multiple roles. An agent may develop a strategy of knowledge acquisition so that it can improve its task reallocations when acting as a relay in a task path. Whereas an agent that is mainly an executor in multiple task paths may ignore system exploration actions and focus purely on optimising its atomic task completion. Agents that perform different roles in multiple task paths need a strategy that is more generalist, taking actions that improve their knowledge of agents in the system, optimise allocation of tasks, and the execution of tasks (see Figure 11.3). Not all strategies are reliant on being part of a task path, e.g. an agent may also prioritise acquiring knowledge so that it can provide a knowledge-sharing service to other agents so that those agents can request information from it, and in so doing, learn to improve their task allocations.

Specialisms

We refer to these strategies as *specialisms*, which are distinct from roles, where certain groups of actions are prioritised by agents over others in order to receive rewards, either through improving their performance as part of a task path, or providing a service to agents that do. We can classify specialisms into three broad categories[6];

[6] Note that specialisms are in reality a continuous spectrum rather than being distinct.

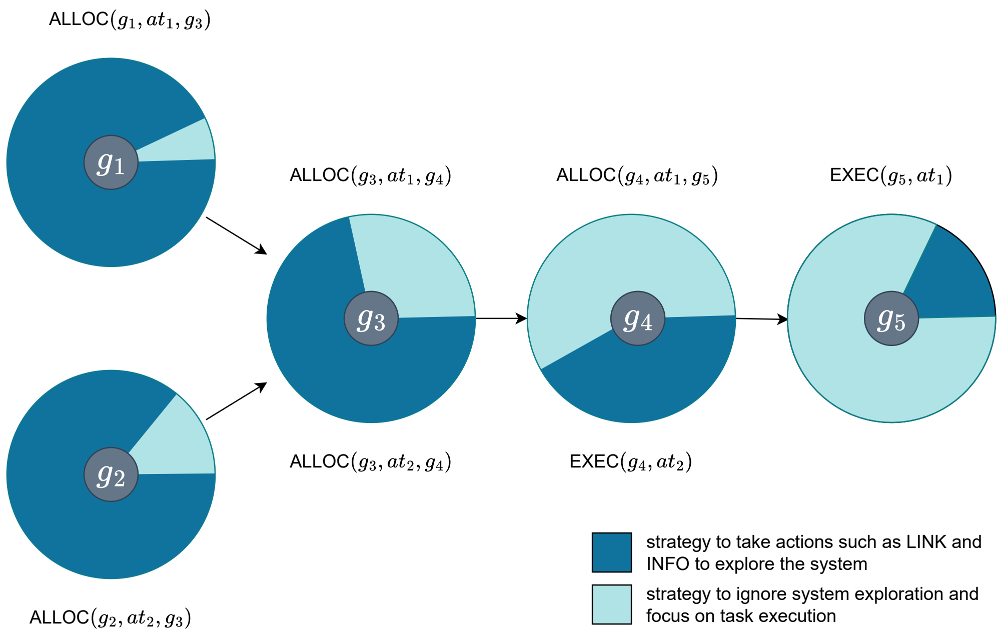


Figure 11.3: Roles and specialisms - agent strategies. *This diagram shows two task paths, where* $path(at_1) = \langle g_1, g_3, g_4, g_5 \rangle$ *and* $path(at_2) = \langle g_2, g_3, g_4 \rangle$. *Agents* g_1 *and* g_2 *assume the role of sinks in both task paths, so their strategies focus on taking actions to learn improvements to their neighbourhoods and knowledge, so they can better allocate their atomic tasks. Agent* g_3 *acts as a relay for both atomic tasks, so has also developed a strategy to learn better neighbourhoods and knowledge with which to provide a better relaying service to agents* g_1 *and* g_2. *Agent* g_4 *is a relay for atomic task* at_1 *and an executor for atomic task* at_2, *and so balances its strategy between task execution, optimising its neighbourhood for allocation, and both relaying and providing knowledge to* g_3. *Agent* g_5 *is an executor for both* at_1 *and* at_2 *and so it carries out few actions to explore possible neighbourhoods and knowledge in the system and primarily focuses on task execution.*

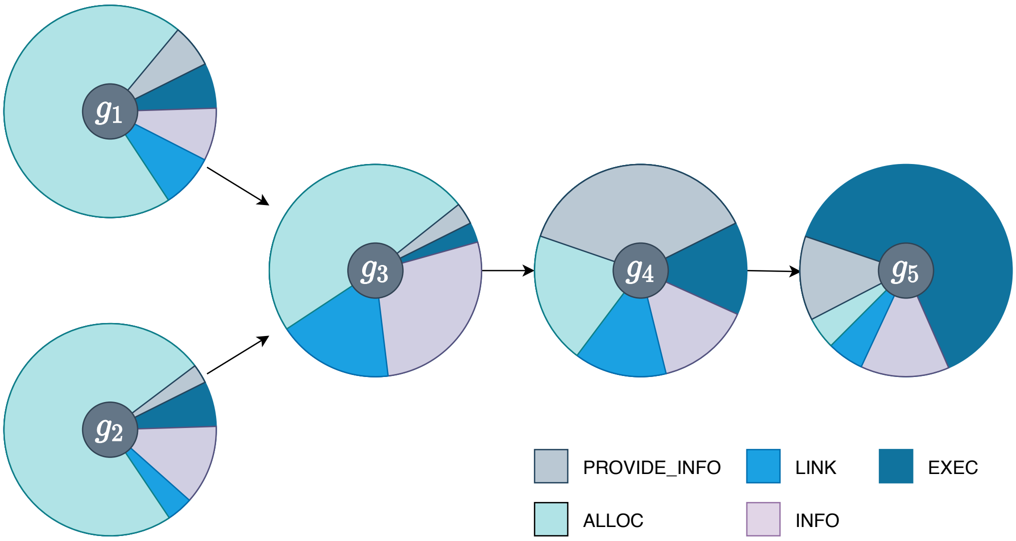


Figure 11.4: Roles and specialisms - agent actions. *This figure shows how agents can develop different specialisms based on their roles in multiple task paths (see Figure 11.3). Agents g_1 and g_2 are sink agents and focus on ALLOC actions to allocate their atomic tasks to g_3 (unaware that these tasks are actually being relayed by g_3 to other agents). Agent g_3 reallocates atomic tasks, but also carries out INFO and LINK actions to improve its allocation performance and so, a better relaying service to the sink agents. Agent g_4 has multiple differing roles and so takes a balanced range of actions, with a noticeable percentage of PROVIDE_INFO actions to provide a knowledge-sharing service to g_3 . Agent g_5 is focused on EXEC actions as it acts mainly as an executor of tasks in the system.*

- *information speciality*, the agent focuses on INFO actions, so that it can PROVIDE_INFO to other agents, enabling those agents to explore the system better;
- *allocation speciality*, the agent balances INFO and LINK actions to optimise its allocation of atomic tasks. This is the case if it were acting as a sink, however, this can also be a strategy developed so that the agent can provide a better target for ALLOC actions by other agents, as is the case when acting as a relay;
- *execution speciality*, the agent mainly ignores system exploration actions such as INFO and LINK and instead targets optimisation of atomic task execution;

We illustrate this concept in Figures 11.3 and 11.4, showing how agents might share their time between actions dependent on their speciality.

11.2.4 Restating the system definition

We restate the system, agent state, and system state definitions here that apply to our final solution as described in the following section.

Definition 11.2.2 (*Distributed Task Allocation System (complete)*). A distributed task-allocation system (DTAS) is defined by a tuple $\langle AT, CT, A, G, RP \rangle$ where:

- AT is the set of atomic tasks at (or tasks for short), where each task can be performed by a single agent;

- CT is the set of composite tasks ct, where each composite task is formed by a set of atomic tasks;
- A is the set of actions that agents can perform;
- G is the set of agents, where each agent g is defined by a tuple $\langle id, c, r, \delta_k, \delta_n \rangle$;
- RP is a set of resource types needed to perform tasks.

See Chapter 8, Section 8.2 for base system definitions, and Chapter 10, Section 10.2 for the definition of resource types RP.

Definition 11.2.3 (*Agent State (complete)*)**.** Given an agent $g = \langle id, c, r, \delta_k, \delta_n \rangle$, we define its state as a tuple $\langle K, N, SP, \Lambda, Q, \mathbf{PW}, \mathbf{E} \rangle$, where:

- $K \subseteq G$ is the knowledge of the agent.
- $N \subset K$ is the neighbourhood of the agent.
- SP is the set of action samples of the agent.
- Λ if the TSQM of the agent.
- Q is the Q-table of the agent.
- $\mathbf{PW}$ is the parent group weights matrix of the agent.
- $\mathbf{E}$ is the eligibility trace matrix of the agent.

Chapter 8, Section 8.3 contains the base agent state definitions. See Chapter 9, Section 9.4.2 for the definition of action samples SP, Section 9.4.5 for the TSQM Λ, and Section 9.4.1 for the Q-table definition. See also Chapter 10, Definition 10.3.3 for the definition of parent group weight matrices $\mathbf{PW}$, and Definition 10.3.7 for eligibility traces $\mathbf{E}$.

Definition 11.2.4 (*System State (complete)*)**.** Given a DTAS we define its state as a tuple $S = \langle G_S, CL, AL, ACT, RAV, RAL, \phi \rangle$ where;

- G_S is the set of states of all agents in the system;
- CL is the set of composite allocations in the system.
- AL is the set of atomic allocations in the system.
- ACT is the set of actions allocated in the system.
- RAV is the set of resource availabilities in the system.
- RAL is the set of resource allocations in the system.
- $\phi \in \mathbb{N}_0$ is the current system counter.

Chapter 8, Section 8.3 contains the base system state definitions. See also Chapter 10, Section 10.2 for the definition of resource availability RAV and allocations RAL.

11.3 The hierarchical task allocation optimisation (HTAO) algorithm

Integrating task paths

The HTAO algorithm integrates the ATA-RIA algorithm from Chapter 9, with the MG-RAO algorithms of Chapter 10 to optimise the utility of the system as described in Equation 10.2.8 (See Chapter 10, Section 10.2). The ATA-RIA algorithm is split into two as compared to Chapter 9 in order to orchestrate the flow of the HTAO algorithm;

- the ATA-RIA (select) algorithm selects actions for agents (See Algorithm 8);
- the ATA-RIA (update) updates the action samples, Q-tables, and TSQM values of agents after they have taken actions (See Algorithm 9).

Note that the combination of the ATA-RIA (select) and ATA-RIA (update) algorithms is equivalent to the ATA-RIA algorithm of Chapter 9, Section 9.5.

The HTAO algorithm

To successfully complete a composite task, a sink must decompose the composite task it received from an external source into atomic tasks. It then selects whether to execute the tasks itself, or allocate them to further agents (See Algorithm 7).

1. initially, the sink receives a composite task ct comprising of a set of atomic tasks to be completed.
2. to track completion and the outputs of tasks throughout the algorithm we list the atomic tasks that are yet to be performed [Line 1] and use a store for the state changes and agent state updates resulting from the agent g taking actions [Line 2].
3. while there are atomic tasks not yet completed or allocated, the ATA-RIA algorithm runs to select an action for the agent g [Line 4];
   (a) if the action chosen is for the agent to execute the atomic task itself, $EXEC(g, at)$, the atomic task is removed from the list of tasks to be completed by the agent [Line 6], and the results of ATA-RIA (select) stored [Line 7].
   (b) if the action chosen is for the agent g to allocate the atomic task, the $ALLOC(g, at, n)$ action, then at is removed from the list of tasks to be completed by the agent [Line 9], and the results of ATA-RIA (select) stored [Line 10].
   (c) if a $LINK(g, k)$ or $INFO(g, n)$ action is executed, these will update the agent's neighbourhood and knowledge base respectively using the ATA-RIA algorithm. There is no effect on atomic task execution or allocation in that case, and the algorithm runs ATA-RIA (update) using the rewards for those actions [Line 13], then loops round to choose another action.
4. the selection and execution of actions using the ATA-RIA (select) algorithm is repeated until the tasks are either executed or allocated. Once this is done, the agent waits for them to complete [Line 16].
5. for each of the atomic tasks that comprise the composite task ct, if the agent is the sink agent for that atomic task, the following steps are performed to allocate rewards [Line 19];

(a) when all the atomic tasks in the composite task have completed, the absolute value[7] for each atomic task is calculated [Line 20]. The agents in each atomic task's task-path are updated using ATA-RIA (update) with these values as rewards for the actions they took in completing the atomic task [Line 23].

(b) finally, for each atomic task, and for each of the resources agent g allocates for task completion, a corresponding reward value (derived from the absolute value of component tasks) is used to update the executor agent's resource weighting model using the MG-RAO (update) algorithm [Line 26], and the new weightings applied to the resource allocation of the system, RAL, using the MG-RAO (weight) algorithm [Line 27].

[7] See Chapter 10, Definition 10.2.7.

ALGORITHM 7: The HTAO algorithm

Input: g, an agent allocated the composite task ct.
Input: ct, a composite task allocated to agent g.
Input: W, the potential change on neighbourhoods on taking an action.
Input: $\langle K, N, SP, \Lambda, Q, \mathbf{PW}, \mathbf{E} \rangle$, the agent state of agent g.
Output: $\langle K', N', SP', \Lambda', Q', \mathbf{PW}', \mathbf{E}' \rangle$, updates to the agent state of agent g.

```
// List atomic tasks to be performed
$ctactive \longleftarrow ct$
// Save the state sequence of all selected actions of tuples.
$CUR \longleftarrow \emptyset$
foreach $at \in ctactive$ do
    // Select and execute action through ATA-RIA
    $\langle g, a, oldstate, newstate, \langle K', SP', Q' \rangle \rangle \longleftarrow$ ataria-select$(g, at, W, \langle K, N, SP, Q \rangle)$
    if $a = EXEC(g, at)$ then
        // Remove the atomic task from the list of atomic tasks to be performed
        $ctactive \longleftarrow ctactive \setminus \{at\}$
        // Store the state change and agent state updates
        $CUR \longleftarrow CUR \cup \langle g, a, oldstate, newstate, \langle K', SP', Q' \rangle \rangle$
    else if $a = ALLOC(g, at, g')$ then
        // Remove the atomic task from the list of atomic tasks to be performed
        $ctactive \longleftarrow ctactive \setminus \{at\}$
        // Store the state change and agent state updates
        $CUR \longleftarrow CUR \cup \langle g, a, oldstate, newstate, \langle K', SP', Q' \rangle \rangle$
    else
        // Update $g$ with rewards immediately non-EXEC or ALLOC actions
        $reward \longleftarrow reward(a)$
        $\langle SP', Q', \Lambda' \rangle \longleftarrow$ ataria-update$(g, a, reward, oldstate, newstate, \langle SP, Q, \Lambda \rangle)$
end
// Wait for all the atomic tasks to be completed
while $\neg complete(ct)$ do
    $wait(g)$
end
foreach $at \in ct$ do
    // Run reward updates only if this agent is a sink agent
    if $g = sink(at)$ then
        // Calculate each atomic tasks' absolute task value
        $taskval \longleftarrow absval(ct, at, AL, RAL)$
        // Calculate a simple reward based on a fraction of the composite task
           reward
        $reward \longleftarrow taskval/|ct|$
        // Send a proportion of the component task value to each task-path agent
        foreach $\langle g, a, oldstate, newstate, \langle K'', SP'', Q'' \rangle \rangle \in CUR$ do
            // Update the Q-values for ALLOC or EXEC actions taken by agents
            $\langle SP', Q', \Lambda' \rangle \longleftarrow$ ataria-update$(g, a, reward, oldstate, newstate, \langle SP'', Q'', \Lambda'' \rangle)$
        end
        foreach $res \in RES$ do
            // Run the MGRAO update for the agent that completed the $at$
            $\langle \mathbf{PW}', \mathbf{E}' \rangle \longleftarrow$ mgrao-update$(executor(at), at, res, taskval, \langle \mathbf{PW}, \mathbf{E} \rangle)$
            // Run the MGRAO weighting
            mgrao-weight$(at, executor(at), pg, res, RAL, \langle \mathbf{PW}, \mathbf{E} \rangle)$
        end
end
return $\langle K', N', SP', \Lambda', Q', \boldsymbol{PW}', \boldsymbol{E}' \rangle$
```

ALGORITHM 8: The ATA-RIA (select) algorithm

Input: g, an agent allocated an atomic task at

Input: at, an atomic task allocated to agent g.

Input: W, the potential change on neighbourhoods on taking an action.

Input: $\langle K, N, SP, Q\rangle$, agent state knowledge, neighbourhood, actions samples, and Q-table of agent g.

Output: g, the agent that selected the action.

Output: a, the action that was selected by the agent g.

Output: $oldstate$, the previous state of the system before action a was selected.

Output: $newstate$, the new state of the system after action a was selected.

Output: $\langle K^{'}, SP^{'}, Q^{'}\rangle$, the updated agent state knowledge, neighbourhood, and actions samples of agent g.

```
// Store current system state
$oldstate \leftarrow s$
// Execute atomic task if agent has capabilities
if $type_a(at) \in c(g)$ then
    $EXEC(g, at)$
    while $\neg complete(\{at\})$ do
        $wait(g)$
    end
    $ct \leftarrow ct \setminus \{at\}$
else
    // Select an action given system state
    $a \leftarrow$ RT-ARP$(g, W, Q)$
    if $a = ALLOC(g, at, n)$ then
        $ALLOC(g, at, n)$
        while $\neg complete(\{at\})$ do
            $wait(g)$
        end
        $ct \leftarrow ct \setminus \{at\}$
    else if $a = INFO(g, n)$ then
        // Get new agent k from action
        $k \leftarrow INFO(g, n)$
        $K^{'} \leftarrow K \cup \{k\}$
        // Prune knowledge base
        $\langle K^{'}, SP^{'}, Q^{'}\rangle \leftarrow$ SAS-KR$(g, \langle K, N, SP, Q\rangle)$
    else if $a = LINK(g, k)$ then
        // Add new agent to neighbourhood
        $LINK(g, k)$
        // Prune neighbourhood based on resources
        $N^{'} \leftarrow$ N-Prune$(g, \langle N, SP\rangle)$
end
// Store new system state
$newstate \leftarrow s$
return $\langle g, a, oldstate, newstate, \langle K^{'}, SP^{'}, Q^{'}\rangle\rangle$
```

ALGORITHM 9: The ATA-RIA (update) algorithm

```
Input: $g$, an agent to be updated.
Input: $a$, the action taken by agent $g$.
Input: $reward$, the reward for agent $g$ as a result of action $a$.
Input: $oldstate$, the state the agent was in when it took action $a$.
Input: $newstate$ , the state the agent was in after it took action $a$.
Input: $\langle SP, Q, \Lambda \rangle$, the agent state of action samples, Q-table, and TSQM, of the agent $g$.
Output: $\langle SP^{'}, Q^{'}, \Lambda^{'} \rangle$, updates to the action samples, Q-table, and TSQM, of the agent $g$.

// Update Q-value mappings using reward generated by action
$Q^{'} \leftarrow rlupdate(g, oldstate, newstate, a, reward, Q)$
// Use the reward value to update the TSQM
$\Lambda^{'} \leftarrow updatetsqm(\Lambda, reward)$
// Update action samples
$SP^{'} \leftarrow SP \cup \{(a, \phi, reward)\}$
return $\langle SP^{'}, Q^{'}, \Lambda^{'} \rangle$
```

11.4 Optimisation using HTAO

The HTAO algorithm optimises performance in 3 main ways;

1. *selecting actions*, when agents have a composite or atomic task allocated to them, they can choose from a range of actions. By utilising reinforcement learning, with the component task values of atomic tasks as rewards[8], HTAO adapts the probability of agents' taking actions to optimise these values;
2. *resource allocation*, as an agent completes an atomic task it will use some resources to do so. The HTAO algorithm predicts the agent's optimal allocation of these resources, given the different atomic tasks it is allocated, and their distribution over time. This allows it to complete the incoming tasks to obtain the best atomic task values overall;
3. *forming task-paths*, the algorithm allows agents to reallocate atomic tasks to other agents, relaying them through the network. It also distributes the reward for completing these tasks across agents in these task paths to optimise the actions that were taken by all agents that participated in the task.

The high-level flowchart in Figure 11.5 shows how sinks decompose composite tasks, choose actions to take, allocate atomic tasks, update Q-tables, updates resource allocation weights and applies them. An example of how the algorithm is applied to a network of agents executing an atomic task is shown in Figure 11.6.

[8] See Chapter 10, Section 10.2, Definition 10.2.7.

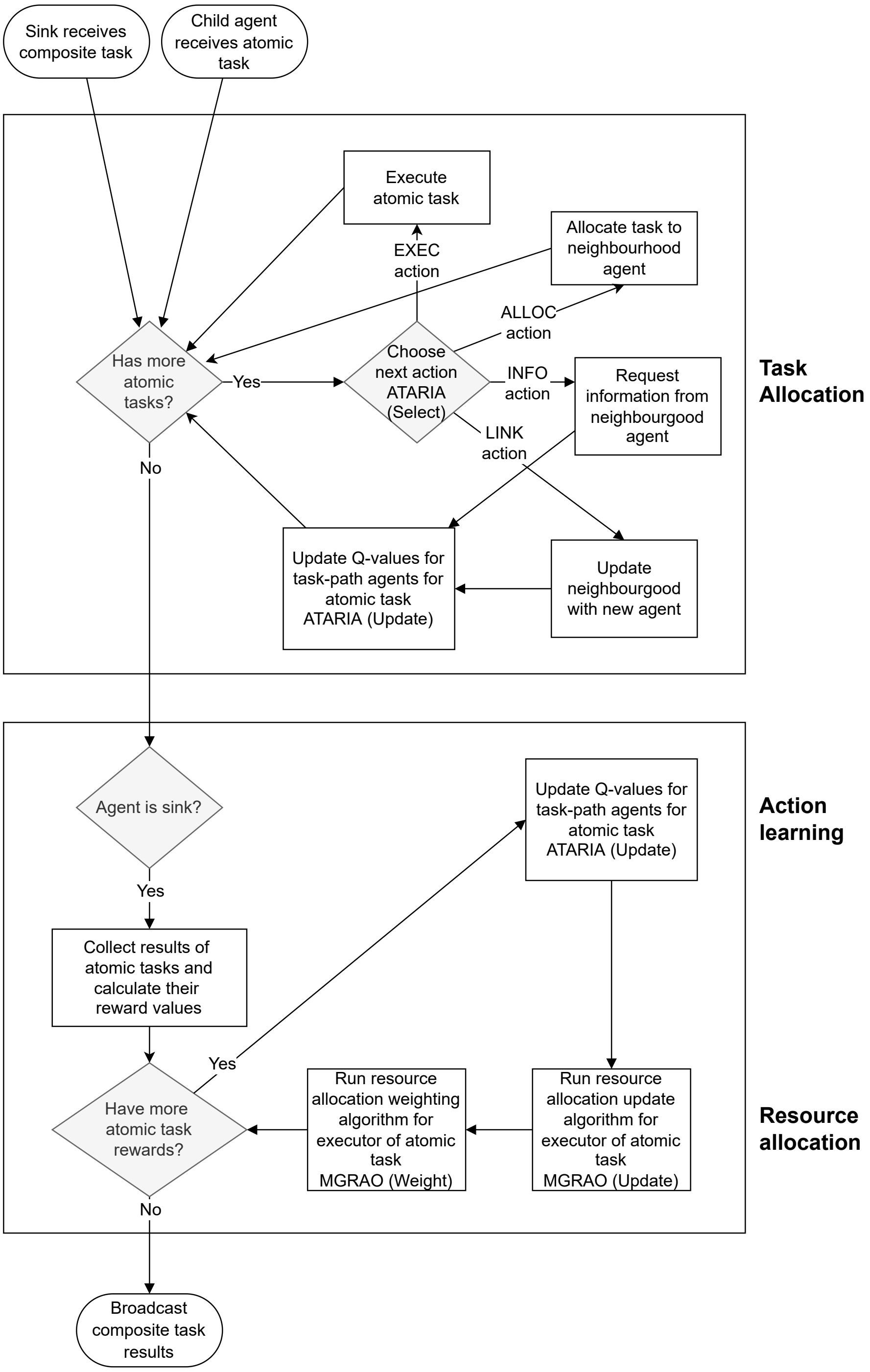
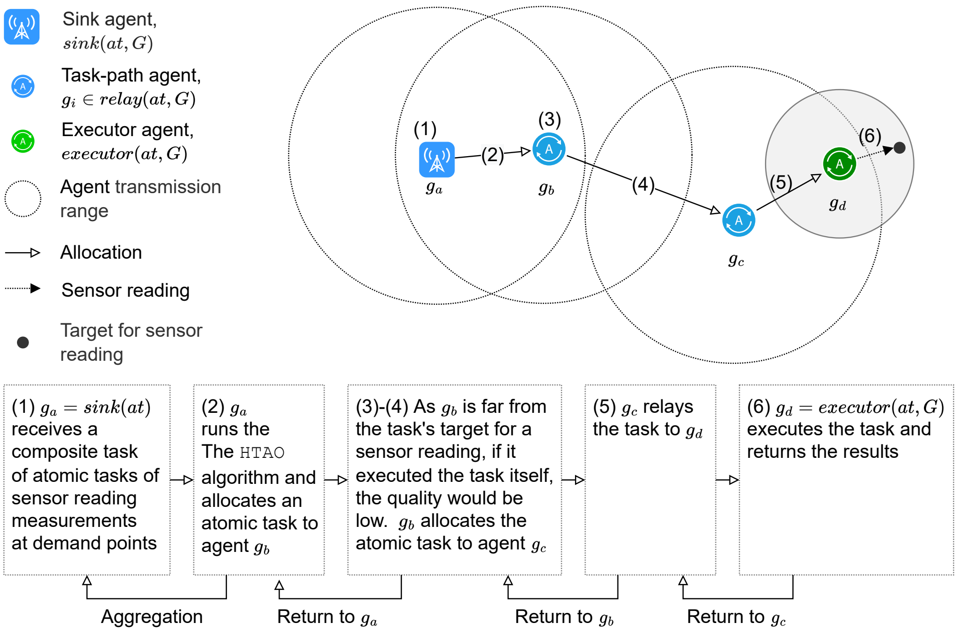

Sink receives composite task
Child agent receives atomic task
Execute atomic task
Allocate task to neighbourhood agent
EXEC action
ALLOC action
Has more atomic tasks?
Yes
Choose next action ATARIA (Select)
INFO action
Request information from neighbourgood agent
Task Allocation
LINK action
No
Update Q-values for task-path agents for atomic task ATARIA (Update)
Update neighbourgood with new agent
Agent is sink?
Update Q-values for task-path agents for atomic task ATARIA (Update)
Action learning
Yes
Collect results of atomic tasks and calculate their reward values
Yes
Have more atomic task rewards?
Run resource allocation weighting algorithm for executor of atomic task MGRAO (Weight)
Run resource allocation update algorithm for executor of atomic task MGRAO (Update)
Resource allocation
No
Broadcast composite task results

Figure 11.5: HTAO execution flowchart. *Shows the flow of execution for the HTAO algorithm, and how it utilises the ATA-RIA and MG-RAO algorithms.*



(1) $g_a = sink(at)$ receives a composite task of atomic tasks of sensor reading measurements at demand points

(2) g_a runs the The HTAO algorithm and allocates an atomic task to agent g_b

(3)-(4) As g_b is far from the task's target for a sensor reading, if it executed the task itself, the quality would be low. g_b allocates the atomic task to agent g_c

(5) g_c relays the task to g_d

(6) $g_d = executor(at, G)$ executes the task and returns the results

Figure 11.6: Allocation along a task-path. *An agent allocates tasks composed of sensor readings at specified locations using the HTAO algorithm. The example task-path has two re-allocations before the specific atomic task is allocated to an agent that completes the task by taking a measurement, then returns the results back along the task path.*

11.5 Summary

In this chapter we brought together the work of Chapters 9 and 10 and integrated the previous ATA-RIA and MG-RAO algorithms to form part of our HTAO algorithm, our holistic solution to the research challenges described in Chapter 1. Using ATA-RIA solves for the task allocation side of the problem, while MG-RAO solves for the resource allocation side. Additionally, we added task paths, allowing agents to reallocate atomic tasks. This adds fault tolerance to the algorithm, as well as an additional route for task allocation optimisation. With the definition of task path roles, we now also have the possibility of agents assuming multiple different roles in completing atomic tasks, the aggregation of these roles giving the agent a specialism, a set of behaviours learned by the agent in order to optimise the utility of the system.

In the next chapter we take the HTAO algorithm and apply it to a WSN simulation. In doing so we evaluate the effectiveness of the algorithm and its behaviour under different conditions.

Chapter 12

Case study: wireless sensor networks (WSN)

In this chapter we apply our HTAO algorithm from Chapter 11 to a simulated environmental monitoring WSN. This enables us to see how well our solution performs in a realistic scenario with the associated complexities, noise, and dynamism that are typical in practical applications.

12.1 Introduction

In Chapter 7 we described some of the applications of WSNs, as well as the challenges they face. In this chapter we use the hierarchical task allocation optimisation (HTAO) algorithm from Chapter 11 to optimise WSN systems based on maximising energy availability, distribution, and task quality, while maintaining task coverage in a dynamic network. We evaluate the algorithm's performance in a simulated environmental monitoring system where there are a number of measurement tasks to be completed.

Chapter structure

In Section 12.2 we cover the high level goals of the WSN. Section 12.3 introduces the common components and resources in our WSN, and how they relate to the concepts described in previous chapters. Section 12.4 focuses on defining the domain-specific components of allocation quality in a WSN. We establish some metrics to measure performance of our algorithms in Section 12.6, which we use in our evaluation of our algorithm in the simulated WSN in Section 12.7, alongside discussion of the results.

12.2 Goals for the WSN system

The goals of the WSN system, as covered in detail in Chapter 7, Section 7.6, are;

1. *energy consumption*, to minimise the energy consumption of the network;
2. *quality of measurement*, to obtain the highest quality of sensor measurements;
3. *sensor coverage*, to obtain the best measurement coverage;
4. *network resilience*, to maintain system performance when there are component failures or environmental effects that reduce agents' abilities to execute tasks;
5. *system lifetime*, to prolong the time in which the system operates within acceptable performance bounds.

We will use these goals to shape the quality of composite tasks, which define the utility of the system. These goals also guide us in specifying metrics used to study the performance of the HTAO algorithm.

12.3 Components, resources, and actions in a WSN

Agents and nodes A WSN consists of a set of interconnected hardware *nodes*. Each node is equipped with the following components[330];

1. a microcontroller for computation;
2. memory storage for holding the results of measurements and knowledge about the system;
3. battery storage, providing power for operational functions, sensor measurements, and communication;
4. sensors for measuring properties of the environment such as temperature or radiation levels;
5. a solar panel for recharging the battery;
6. a wireless transceiver for transmitting and receiving messages from other nodes.

Each node has an agent g, a software controller that instructs its actions. For simplicity, we will use the term 'agent' to refer to both the software controller and the hardware node it controls.

Example 12.3.1 (*Example of an ocean-based WSN*)**.** An ocean-monitoring WSN system has 100 agents, each with a sensor to measure salinity, deployed into an ocean bay 1 km^2 in area. Each agent is attached to a buoy to maintain its position. The agents each have availability of multiple resource types; energy rp_{pow}, compute rp_{cpu}, and memory rp_{mem}. To meet its performance goals, the system needs to;

- form and maintain an ad-hoc communication network, adaptable to changes in the environment.
- return the most accurate results, from the desired locations, every hour to the base station;

- minimise the energy used by agents so that their batteries can charge enough between repeated requests, in order to maintain coverage;
- distribute the tasks so that there is broad sensor coverage for the measurement tasks;
- distribute the energy usage so that the wear and subsequent failure of agents is reduced, meaning the system's lifetime exceeds the operational design of 6 months.

Every hour, an agent acting as a sink receives a composite task from an on-shore base station to measure the salinity in 10 specific locations, or demand points. The sink has knowledge of another 20 agents in the system that can help it with its task, however, it has only established communication links with 5 of these agents, which is its neighbourhood. The sink decomposes the composite task into atomic tasks $\{at_i\}_{i=1}^{10}$, then allocates these tasks to other agents in its neighbourhood.

Executor agents use their sensors to take the measurements, and return the results to the agent that allocated them the task, until it reaches the sink. The sink agent then aggregates the data and broadcasts it back to the base station.

Resources

To complete tasks WSN agents will need to allocate resources, of one of those shown below (alongside their main use cases);

- *compute resources*, used in general calculations, activating sensors and processing readings. Each agent has the same fixed amount of compute resource, which it must share amongst its current tasks;
- *memory resources*, needed to store knowledge about other agents, network routes, and the results of tasks. Each agent has the same fixed amount of this resource. This restricts the amount of knowledge of the system that an agent can have at any one time;
- *energy resources*, required to transmit and receive task requests, results, and knowledge between agents, as well as in the utilisation of sensors for taking measurements. Each agent has the same, fixed battery capacity at system initialisation. Each action uses some of this energy, which is gradually replenished by a solar panel.

We categorise resource usage by the following criteria[1];

1. *operational cost*, resource usage resulting from an agent's general operations that are not part of a task execution (e.g. using energy moving between idle and sleep modes[331], or the computational and memory storage costs of background functions);
2. *dominant cost*, resource usage that is significantly larger than others in comparison (e.g. when completing tasks, the energy used to transmit and receive results is much larger than the energy used to construct the message);
3. *optimisable cost*, resource usage that can be actively optimised by the agent,

[1] For many systems the simplifications discussed are reasonable, but not for all. For example, some sensors may have significant energy requirements beyond their activation costs. This increases the complexity of the modelling but can be incorporated into the same framework we present.

and so can be used by the agent to learn new strategies (e.g. agents could communicate with other agents to which there is a shorter transmission range, or reduce the sampling time of a sensor to use less energy).

We focus on modelling dominant and optimisable costs in our WSN simulation, ignoring the complexities that would be introduced through operational costs. There exists other work that covers this area such as on the topics of battery duty-cycling[332], transmission algorithms[333], [334], and solar-energy harvesting optimisation[335], [336].

Modelling actions in a WSN

With the justification on resource costs described above, we give examples of the types of actions agents might take in a WSN, and the resultant resource usage, in Table 12.1.

Action	**Example**	**Resource usage**
ASSIGN	a ground station broadcasts a request to buoys holding sensors in an ocean monitoring WSN[337].	computational and memory resources are used by the agent receiving the request to store the composite task and aggregated results.
EXEC	an agent takes a temperature sensor measurement in its location.	the agent can adapt the amount of computational resource allocated to this type of task, increasing their quality the more that it allocates.
ALLOC	the agent allocates a measurement task to another agent. Allocating to nearer agents can reduce energy use but may increase the distance from the sensor to the desired target location for the measurement.	this uses energy resources to transmit tasks and receive results, increasing with the distance between the two agents.
INFO	the agent asks for routes to other agents from another agent. New routes could give higher quality sensor readings, or access to geographical areas that were previously unreachable.	this uses energy to transmit and receive messages.
PROVIDE_INFO	an agent returns knowledge of other agents in the system.	uses memory storage for the knowledge received.
REMOVE_INFO	an agent deletes knowledge about network routes to reach agents that it considers less useful to improving its performance.	frees up memory storage that can be used for new knowledge.
LINK	an agent carrying out a link action will instantiate a connection to another agent it knows of, allowing it to send and receive messages to that agent.	computational resource and memory storage are allocated to maintain the connection.

continues on the next page...

...continued from previous page

REMOVE_LINK	an agent disables keep-alive signalling to another agent agent, an removes it from route calculation and task allocation selection[338]	CPU resource usage is reduced as less routes are searched and allocation calculations required for each atomic task allocated.

Table 12.1: Modelling actions in a WSN. *Examples of actions that agents in a WSN could take, and the corresponding effects of resource management*

12.4 The allocation quality of sensor measurement tasks

We now derive the domain-specific allocation quality, driven by the desired goals described for the WSN system in Section 12.2.

Geography

The WSN is deployed over a finite geographical area, with agents arbitrarily located within this bounded area (as occurs in an aerial deployment[339]), where they form an ad-hoc communication network[277]. This area is overlaid by a 2-dimensional Cartesian coordinate system, $\mathbb{R} \times \mathbb{R}$ that covers all agents and possible measurement locations in the system.

Deployment

An agent's *deployment* is its mapping to a specific (x, y) location in this coordinate system; $deploy\colon \mathcal{G} \rightarrow \mathbb{R} \times \mathbb{R}$. To simplify calculations, the coordinate system is assumed unit-normalised (i.e. $\mathbb{R}[0,1] \times \mathbb{R}[0,1]$), so the maximum distance between any two locations is $\sqrt{2}$.

Demand points

We assume that each atomic task to take a sensor measurement within a geographically based WSN, such as the one in this case study, specifies a *demand point*[340], [341], a specific location targeted for that measurement: $demand\colon \mathcal{AT} \rightarrow 2^{\mathbb{R}\times\mathbb{R}}$.

In the context of our WSN case study, the further away an agent is from an atomic task's demand point, the less value completing an atomic task has to the system. E.g. an agent taking a measurement of ocean temperature when 100 metres away from the demand point of the corresponding task may result in a very different reading than the actual value at the targeted point.

Execution range

To capture this, we add a dependency to the allocation quality of atomic tasks[2], the *execution range* of an atomic task, which measures the distance between an executor agent g and the demand point of the atomic task at that it is completing, $range\colon \mathcal{AT} \times \mathcal{G} \rightarrow \mathbb{R}$, where:

$$range(at, g) = 1 - \frac{|demand(at) - deploy(g)|}{\sqrt{2}} \tag{12.1}$$

As we specify a unit-normalised coordinate system, the maximum distance between any agent this point is $\sqrt{2}$, a value of 1 would mean that the agent is at the demand point of at, decreasing towards 0 as the separation increases [3].

[2] See Chapter 10, Section 10.2.

[3] We assume a linear drop-off in quality, but this is not a requirement and more complex modelling could be used.

Example 12.4.1 (*Demand points and allocation quality*). Two identical agents g_1 and g_2 are allocated an atomic task at to take a salinity measurement in an ocean-monitoring system, where g_1 is much closer to the demand point of at than g_2 (i.e. $|demand(at) - deploy(g_1)| \ll |demand(at) - deploy(g_2)|$). Since $range(at_1, g)$ is then closer to 1 than $range(at_2, g)$, all other considerations being equal, the allocation of at to g_1 would give the higher allocation quality than to g_2.

12.5 Energy availability, distribution, and costs in a task path

In order to define the domain-specific allocation quality components in a way that will meet the WSN goals of Section 12.2 we use three factors;

1. *average resource availability*, by increasing this value, the system will be be less likely to become optimised towards using focusing resource usage on a small subset of agents in the system (See Figure 12.1);
2. *resource usage distribution*, increasing this value spreads resource usage more equally amongst agents, improving resilience, and system lifetime by reducing wear (See Figure 12.2);
3. *task path cost*, by decreasing the cost of the path used to complete an atomic task, less resources will be used in each task completion.

Average resource availability

We define the average resource availability of a resource type rp amongst a group of agents G, $resaverage \colon 2^{\mathcal{G}} \times \mathcal{RP} \times \mathcal{RAV} \to \mathbb{R}$:

$$resaverage(G, rp, RAV) = \frac{1}{|G|} \sum_{\forall g \in path(at)} resavailable(g, type_r(res), RAV) \qquad (12.2)$$

Resource distribution

The distribution of a resource of type rp across a group of agents G can be measured using the variance of the set of the individual resource availabilities of the agents[4]. Given a maximum possible availability of a resource type of value $maxres_{rp}$ (e.g. the capacity of a battery), we can reshape this as the distance between the variance and the maximum bound $\frac{maxres_{rp}}{4}$, so distribution can be optimised by maximisation of the function. This gives us the *resource distribution* across a group of agents G, $resdist \colon 2^{\mathcal{G}} \times \mathcal{RP} \times \mathcal{RAV} \to \mathbb{R}$:

$$resdist(G, rp, RAV) = \frac{maxres^2_{rp}}{4} - \sigma^2\left(\left\{resavailable(g, rp, RAV)\right\}_{\forall g \in G}\right) \qquad (12.3)$$

[4] Where we use the standard definition of variability of a discrete set X, $\sigma^2(X) = \frac{\sum(x_i - \bar{x})^2}{|X|}$.

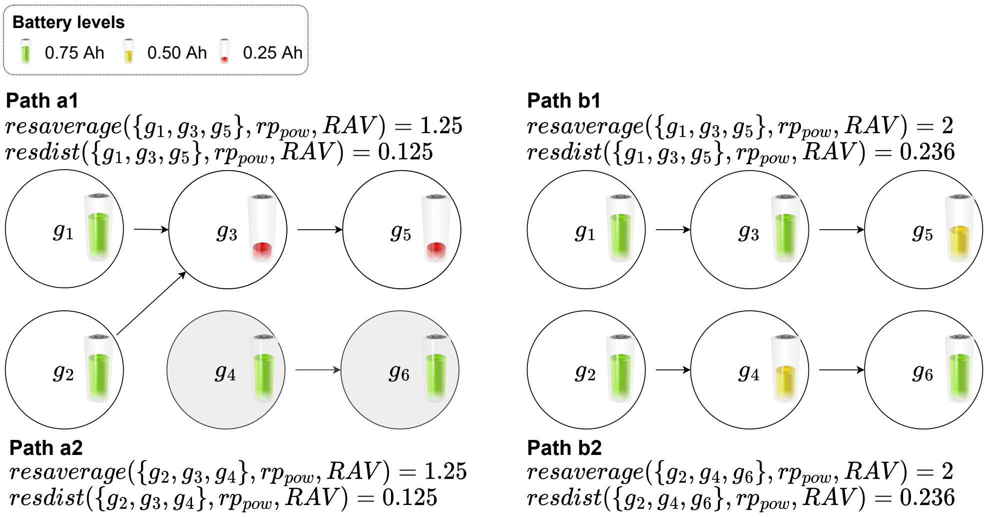

Battery levels
0.75 Ah
0.50 Ah
0.25 Ah
Path a1
$resaverage(\{g_1, g_3, g_5\}, rp_{pow}, RAV) = 1.25$
$resdist(\{g_1, g_3, g_5\}, rp_{pow}, RAV) = 0.125$
Path b1
$resaverage(\{g_1, g_3, g_5\}, rp_{pow}, RAV) = 2$
$resdist(\{g_1, g_3, g_5\}, rp_{pow}, RAV) = 0.236$
g_1
g_3
g_5
g_1
g_3
g_5
g_2
g_4
g_6
g_2
g_4
g_6
Path a2
$resaverage(\{g_2, g_3, g_4\}, rp_{pow}, RAV) = 1.25$
$resdist(\{g_2, g_3, g_4\}, rp_{pow}, RAV) = 0.125$
Path b2
$resaverage(\{g_2, g_4, g_6\}, rp_{pow}, RAV) = 2$
$resdist(\{g_2, g_4, g_6\}, rp_{pow}, RAV) = 0.236$

Figure 12.1: Domain-specific allocation quality - energy availability. *Agents g_1 and g_2 are allocating 2 atomic tasks at_1 and at_2. As the resaverage of the paths b1/b2 is higher than those of the paths a1/a2, the energy usage in the system overall is better distributed if path options like b1/b2 are preferentially chosen over those like a1/a2.*

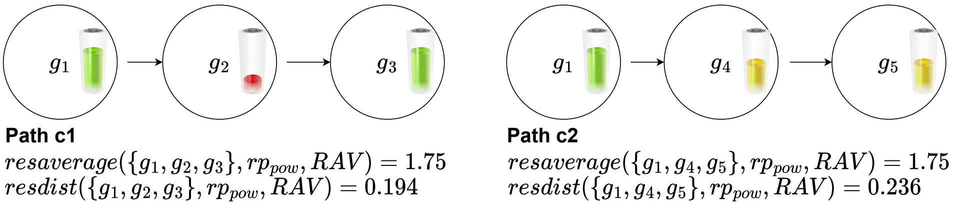

g_1
g_2
g_3
g_1
g_4
g_5
Path c1
$resaverage(\{g_1, g_2, g_3\}, rp_{pow}, RAV) = 1.75$
$resdist(\{g_1, g_2, g_3\}, rp_{pow}, RAV) = 0.194$
Path c2
$resaverage(\{g_1, g_4, g_5\}, rp_{pow}, RAV) = 1.75$
$resdist(\{g_1, g_4, g_5\}, rp_{pow}, RAV) = 0.236$

Figure 12.2: Domain-specific allocation quality - energy distribution. *The paths c1 and c2 have the same resaverage, however, resdist is higher for path c2, meaning the energy distribution is more balanced. Choosing more paths like c2 can reduce the wear on individual agents, and so their failure rate, therefore improving the lifetime of the WSN.*

Example 12.5.1 (*Resource distribution in a WSN*). Agents $G = \{g_1, g_2, g_3\}$ in a system each have a battery with a maximum capacity of $1\,Ah$. Agent g_1 currently has 10% of its energy available to use, g_2 has 90%, and g_3 has 40%, so:

$$resdist(G, rp, RAV) = \frac{1}{4} - \sigma^2\left(\left\{0.10, 0.90, 0.40\right\}\right) \cong 0.14$$

Task path cost in a WSN To calculate the cost of task paths in a WSN we look at the dominant costs involved as a simple first approximation of the overall costs[342];

- *communication cost*, the energy cost to an agent of making an outgoing communication with another agent. We assume this is invariant both across atomic task types (i.e. the energy to transmit an atomic task message to another agent is not dependent on the type of task), as well as the energy used by agents to send a message. We further assume that there is a linear relationship between the distance separating agents and the energy cost, so that in transmitting a message from g_1 to g_2, there is a fixed amount of energy resource res_{tx} such that the communication cost is $res_{tx}|deploy(g_2) - deploy(g_1)|$;
- *execution cost*, the energy cost to an agent of executing an atomic task, invariant for each atomic task type (i.e. the energy used to activate a radiation sensor is the same for all atomic tasks of the type that require that sensor, and no matter which of the agents is operating the sensor)[5]. We make the common assumption that sensor activation is the dominant energy cost when taking measurements, with multiple sampling being less significant[343]. This means that the overall energy use of a sensor while executing a task remains approximately constant for each type of atomic task, and quality of its completion. I.e. $\forall ap \in AP$ there is a fixed amount of energy resource for each atomic task type ap given by $res_{exe}(ap)$.

There are other possible costs such as the cost to process the results of an atomic task execution and assemble a message. We assume these other costs are of negligible impact to the overall cost in comparison to the dominant costs mentioned. Therefore, the task path cost for the WSN system can be approximated by:

$$\begin{aligned} pathcost(at, G, rp_{pow}) = \; & \underbrace{res_{tx}|deploy(g_2) - deploy(g_1)|}_{\text{sink cost}} \\ & + \sum_{i=2}^{n-1} \underbrace{2res_{tx}|deploy(g_{i+1}) - deploy(g_i)|}_{\text{cost per relay agent}} \\ & + \underbrace{res_{tx}|deploy(g_n) - deploy(g_{n-1})| + res_{exe}(type_a(at))}_{\text{executor cost}} \end{aligned} \tag{12.4}$$

where, $sink(at) = g_1$, $relays(at) = \langle g_2 \ldots g_{n-1}\rangle$, $executor(at) = g_n$.

[5] Note, this is a separate and additional resource cost to the resources allocated and utilised by agents to increase atomic task quality as discussed in Chapter 10.

Example 12.5.2 (*Types of energy cost for agents in a WSN*). An agent g_1 allocates atomic tasks at_1 and at_2 to an agent g_2. These tasks are of types ap_{rad} to take a radiation measurement, and ap_{oxy} to measure oxygen level, respectively. The allocation of each task requires the agent g_1 to activate its transceiver and broadcast a message to g_2. The same amount of energy, or *communication cost*, is used no matter the type of the atomic task. To carry out task at_1 the agent g_2 expends energy to activate its radiation sensor. The oxygen sensor required for task at_2 has a much lower energy requirement for activation, therefore the *execution cost* of at_1 is much greater than that of at_2. When measuring radiation levels for at_1, g_2 uses a long sample time to increase the accuracy, therefore increasing that task quality on completion but at the cost of using more of its energy resource, this is its *resource allocation* for tasks of this type.

Example 12.5.3 (*Task path cost in a WSN*). A sink g_1 in an ocean monitoring WSN has been allocated an atomic task at_{sal} to measure the salinity of the water at a specific location. It allocates the task to an agent g_2, which reallocates it to an agent g_3, and then to g_4, which makes the measurement (i.e. $path(at) = \langle g_1, g_2, g_3, g_4 \rangle$). The energy cost of broadcasting a message between g_1 and g_2 is $40mA$, and between g_2 and g_3, $60mA$, and g_3 and g_4, $50mA$. Activating g_3's salinity sensor takes energy, $200mA$. Therefore, the overall task path cost for the energy resource is:

$$\begin{aligned} pathcost(at, path(at), rp_{pow}) &= (2 \times 40) + (2 \times 60) + (2 \times 50) + 200 \\ &= 400mA \end{aligned}$$

Domain-specific allocation quality component (WSN)

We can now expand the domain-specific atomic task and allocation quality components as defined in Chapter 10, Section 10.2 to include the execution range, as well as the average availability, distribution, and task path cost of the energy resource type rp_{pow}:

$$\begin{aligned} domtask(at, g) &= c_{ql}\, range(at, g) \\ domql(at, g) &= c_{pow} \underbrace{resaverage(path(at), rp_{pow}, RAV)}_{\text{average energy available in task path}} \\ &\quad + c_{dst} \underbrace{resdist(path(at), rp_{pow}, RAV)}_{\text{energy distribution across task path}} \\ &\quad + c_{cost} \underbrace{\frac{1}{pathcost(at, path(at), rp_{pow})}}_{\text{energy cost of task path}} \end{aligned} \tag{12.5}$$

Where c_{ql} is a constant chosen at system initialisation to scale the value of atomic task qualities, and c_{cost}, c_{pow} and c_{dst}, are constants to weight the influence of path cost, energy availability, and distribution respectively, depending on the desired properties of the specific system. e.g. if $c_{ql} \gg c_{cost}, c_{pow}, c_{dst}$ then the system will optimise allocation quality for the atomic task quality, to the detriment of energy consumption and usage distribution, with the resulting effect on resilience and lifetime.

12.6 Metrics for WSN analysis

In order to understand the system's performance against the goals of Section 12.2 we develop coverage, resilience, and lifetime metrics in this section. To measure coverage, we must first define what it means for a task to be completed successfully in the context of our WSN.

Atomic task success An atomic task is considered successful if the agent that completes it is within a maximum range, *maxrange* (defined on system initialisation), of the task's demand point, $success\colon \mathcal{AT} \to \mathbb{B}$:

$$success(at) = \begin{cases} 1, & \text{if } range(at, executor(at)) < maxrange \\ 0, & \text{otherwise} \end{cases} \tag{12.6}$$

In many scenarios, when the quality of an atomic task completion falls below a certain threshold the result is no longer be useful or relevant to the system; e.g. when an agent takes a sensor measurement of such low quality that it is masked by background variations in the environment. These scenarios could be simply accounted for through a different domain-specific definition of the success function above.

Coverage Given a completed set of composite tasks, CT, then the *system coverage*, $syscov\colon 2^{\mathcal{CT}} \to \mathbb{R}[0,1]$ is the fraction of those atomic tasks composing the composite tasks which were completed successfully:

$$syscov(CT) = \frac{1}{|CT|} \sum_{\forall ct \in CT} \sum_{\forall at \in ct} \frac{success(at)}{|ct|} \tag{12.7}$$

Example 12.6.1 (*Success and coverage*). An ocean-monitoring system has a deployment area of 1 km^2. A sink agent receives a composite task $ct = \{at_1, at_2\}$ where both the atomic tasks specify taking salinity measurements at different demand points. An agent g is allocated both tasks, and so takes a salinity measurement 1 metre away from a demand point of a task at_1, and 100 metres from that of at_2. The result of at_1 is likely to be close to the actual value at its demand point location, whereas that of at_2 is so far away as to uncorrelated, and not a practically useful measurement to the system. In this case, we might have $maxrange = 10$, so that $success(ct, at_1) = 1$ and $success(ct, at_2) = 0$. The system coverage is then $syscov(\{ct\}) = \frac{1}{2}(1 + 0) = 0.5$

Resilience Agents in a system may become either permanently or temporarily unavailable for the allocation of tasks through events such as component failures, communication problems, or weather disruption. In these circumstances, being able to allocate atomic tasks to agents that can successfully complete them[6] can become impossible. The ability to maintain coverage under these circumstances defines a system's *resilience*.

[6] As in the definition given by Equation 12.6.

Specifically, if a set of composite tasks CT is completed by a set of agents G, where only agents in the set, $G^{'} \subseteq G$ were functional and able to complete atomic tasks, then the resilience of the system is defined by the function $resilience\colon 2^{\mathcal{CT}} \times 2^{\mathcal{G}} \times 2^{\mathcal{G}} \longrightarrow \mathbb{R}$, where:

$$resilience(CT, G^{'}, G) = \frac{syscov(CT)}{|G'|/|G|} \tag{12.8}$$

Therefore, a system whose coverage remains high as the number of available agents falls has a higher resilience that a system whose coverage drops lower under the same circumstances.

Lifetime

As the system's lifespan increases, more agents will become permanently unavailable for allocation (e.g. agents suffering complete sensor or battery failures), and there will be a progressive overall deterioration in coverage. A minimum coverage value $mincov$ for some set of composite tasks CT can be chosen below which the system is judged to be no longer useful. The *lifetime* of the system can then be defined as the time until $syscov(CT) < mincov$.

Example 12.6.2 (*Resilience and lifetime in an ocean WSN*)**.** 100 agents attached to buoys are deployed into an ocean bay to carry out salinity measurement tasks, so $|G| = 100$. Due to rough weather, the sensor equipment on 10 of the agents is non-functional at the time a composite task ct is carried out, so $|G^{'}| = 90$. As a result, 2 of the 10 atomic tasks of a composite task being performed in the system during this period are not successful, due to being executed by agents that are far away from their demand points. Therefore, the resilience is:

$$resilience(\{ct\}, G^{'}, G) = \frac{8/10}{90/100} \approxeq 0.89 \tag{12.9}$$

The system stops being useful when less than 50% of the measurements are successful, i.e. the salinity is not measured at enough locations for useful analysis, $mincov = 0.5$. After 22 weeks, less than 50% of atomic tasks are being completed to a successful value, defining the system's lifetime.

12.7 Evaluation

The simulation framework to evaluate our solution's performance was based on a realistic deployment scenario as covered by[243] and others[254], [344]. In this scenario a UAV is used to deploy numerous sensors over an expansive and remote geographical area, giving an ad-hoc, randomised placement of devices. Solar power cells are used to maintain enough energy to power the agents over a number of months, given low enough power consumption.

Baseline algorithm

For comparison, we use a Q-routing-based algorithm[20], [345], [346], utilising composite task quality from Chapter 10, Section 10.2.6 as a reward function, along with the domain-specific components for allocation quality from Section 12.4. This gives a connectionless, multi-objective variant of Q-routing[7]. Although agents use only local information to make routing decisions, they used full system knowledge and neigh-

[7] See Appendix E.1.

bourhood information, limiting this comparison algorithm's applicability to large systems due to WSN resource limitations[8].

Simulation types We ran 3 simulations to evaluate the performance of our algorithm, with each run replicated 100 times, and the data aggregated for each simulation type. In the first simulation, the stable system, the HTAO algorithm had equal weighting for each of the domain-specific allocation quality components c_{cost}, c_{pow}, c_{dst}, and c_{ql} to examine system utility and energy optimisation in comparison to the Q-routing baseline algorithm. We then looked at the same configuration in the biased system but where the energy, quality, and distribution composite task quality components were each given an 80% dominance over the other components in three separate configurations of the algorithm (see Table 12.2). By evaluating the algorithm in different configurations we could see how adaptable it was to differing system goals such as when more quality, energy-efficiency, or distribution focused. Finally, we looked at the degraded system which simulated system degradation through gradual, permanent loss of agent availability. This studied how the deterioration of agent availability effects optimisation efficiency and the coverage of tasks.

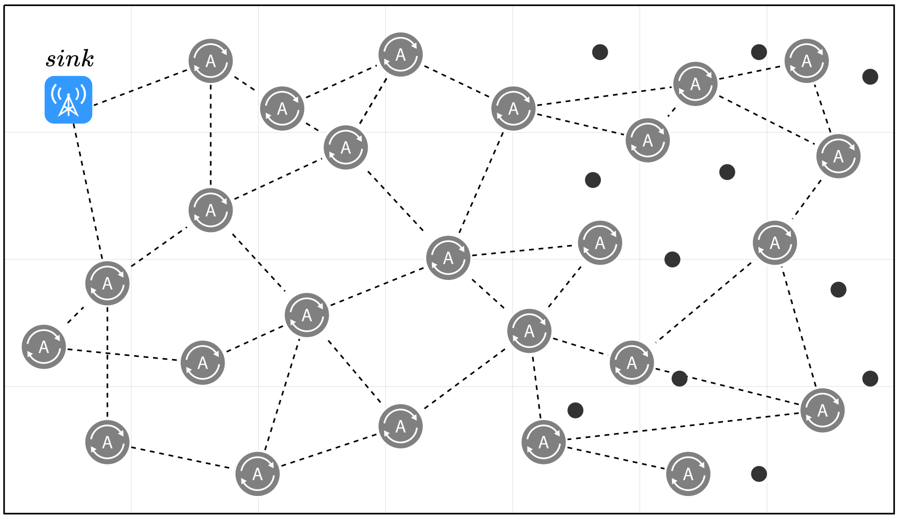
General environment In all systems there was 1 sink and 25 other agents distributed randomly. Each was initially connected to 3 other agents that were randomly chosen with a bias towards nearby agents[9]. The sink was given 3 composite tasks to complete sequentially, each consisting of 10 different atomic tasks for taking sensor measurements. The demand points of the atomic tasks were distributed further away from the sink rather than randomly spread. The completion of the 3 composite tasks marked the end of an episode, and the process was repeated for 1000 episodes. Each non-sink agent was capable of completing an atomic task, or allocating it to any of 3 agents it was connected to. They could complete any measurement task with an allocation quality dependent on their closeness to the demand point associated with the task (see Section 12.4, Eq. 12.5). The energy stored in the batteries of agents in the system was increased by 25% of their capacity (up to the maximum) at the end of each episode, representing solar harvesting. An example of this initial system state can be seen in Figure 12.3. Note, this represents the initial configuration of the system, which is then adapted by agents as they learn to optimise their neighbourhoods.

The degraded system introduced randomised, permanent loss of agent availability over time, showing how each algorithm maintains coverage and optimises for quality and energy while agent availability is lost. The simulation ran for 100 episodes, where for each episode between 20 and 40 a randomly chosen single agent might fail with probability 0.25. Once an agent was unavailable, it remained so for all further episodes. An example of this initial system state is shown in Figure 12.4.

System configurations and results Labels, descriptions, and configurations for each algorithm are shown in Table 12.2, with system constants in Table 12.3. Table 12.4 shows the average and cumulative system utility, and Table 12.5 energy available values, for both algorithms in the stable and degraded systems. For the degraded system, it also includes the percentage drop in these values during the period of agent availability loss. The impact of these failures on the percentage coverage (Section 12.6, Eq. 12.7) for tasks is shown in Table 12.6.

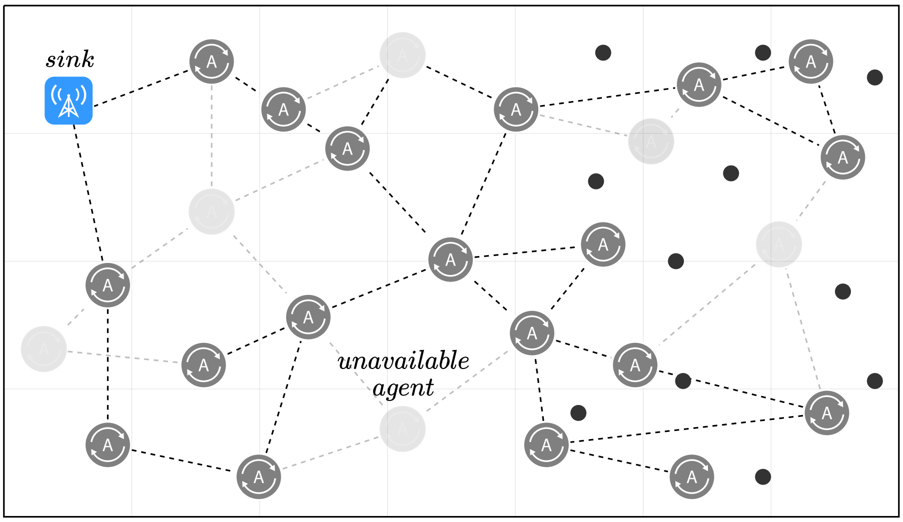
[8] As discussed in Section 12.3.

[9] The probability of initial agent selection used the standard gamma distribution $f(x;\alpha,\beta) = \frac{\beta^{\alpha} x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)}$. Where $\Gamma(\alpha)$ is the gamma function $\alpha = 1.8, \beta = 0.5$, and x is the unit distance between the two agents.



(a) stable system

Figure 12.3: Stable system type. *The diagram shows an example of a* `stable` *system, there are* 25 *agents that can execute the measurement tasks. The tasks' demand points are clustered away from the sink.*



(b) degraded system

Figure 12.4: Degraded system type. *The diagram shows an example of a* `degraded` *system. A number of agents are unavailable, reducing the choice of task-paths as compared to the* `stable` *system of Figure 12.3.*

The results for the htao (energy), htao (quality), htao (distribution) configurations of the HTAO algorithm in the biased system are shown in Table 12.7.

The average utility is shown in Figures 12.5 and 12.11 for the stable and degraded systems with the corresponding cumulative sums in Figures 12.6 and 12.12. For the degraded system we graph the percentage of agents unavailable for task allocations in the system in Figure 12.9 and its effect on percentage coverage (Section 12.6, Eq. 12.7) in Figure 12.10. For the biased system we compare the different biases for optimisation across the composite task quality components using quality-energy balance, task distribution, and task-path depth results. The quality-energy data shown in Figure 12.15 uses the htao (energy) algorithm as a baseline, with the percentage increase or decrease in the average composite task quality over energy availability. The task-distribution in Figure 12.16 shows the variation in the agents that are completing the tasks[10], with higher values representing more tasks being completed by distinct agents, and lower values meaning more agents are completing multiple tasks. Task-path length data in Figure 12.17 captures how many agents recursively allocated each task before it was completed. The p-values showing the statistical significance of the system utility values are shown in Appendix E Table E.1.

Algorithm	**Summary**	**Agent count**	**Atomic tasks**	$(c_{cost}, c_{pow}, c_{dst}, c_{ql})$
htao (stable)	HTAO with balanced objectives.	25	10	$(0.25, 0.25, 0.25, 0.25)$
q-routing (stable)	Q-routing with balanced objectives.	25	10	$(0.25, 0.25, 0.25, 0.25)$
htao (energy)	HTAO with 80% bias for energy consumption minimisation	25	10	$(0.40, 0.40, 0.10, 0.10)$
htao (distribution)	HTAO with 80% bias for energy distribution maximisation.	25	10	$(0.05, 0.05, 0.80, 0.10)$
htao (quality)	HTAO with 80% bias for task quality maximisation.	25	10	$(0.05, 0.05, 0.10, 0.80)$
htao (degraded)	HTAO with balanced objectives.	25	10	$(0.25, 0.25, 0.25, 0.25)$
q-routing (degraded)	Q-routing with balanced objectives.	25	10	$(0.25, 0.25, 0.5, 0.0)$

Table 12.2: WSN: Summary of algorithm and system configurations. *Configurations for the stable, energy, quality, distribution, and degraded systems for the HTAO and Q-routing algorithms.*

[10] Calculated as the fraction of unique sensing agents completing all the atomic tasks during an episode. I.e, if a set of atomic tasks AT is completed by agents $G^{'}$, which may contain duplicates of the same agent completing different tasks, then the fraction of unique agents would be $|set(G^{'})|/|G^{'}|$

Constant	**Summary**	**value**
$maxres_{pow}$	The maximum availability of the energy resource type per-agent.	1.0
$maxrange$	The maximum range between an atomic task's demand point and the location of the agent completing that task for the result to be considered successful in terms of the coverage of the system.	0.25
$mincov$	The minimum coverage of the system before it is considered to have reached the end of its useful lifetime.	0.8

Table 12.3: WSN: Summary of constants. *Fixed constants for the WSN system.*

Algorithm	**Utility (average)**		**Utility (impact)**	**Utility (cumulative)**
	start	**finish**		
htao (stable)	18.2	21.1	n/a	138
q-routing (stable)	18.0	20.8	n/a	0.0
htao (degraded)	17.9	18.3	−2.1%	40
q-routing (degraded)	17.6	17.8	−6.0%	0.0

Table 12.4: WSN: Average and cumulative system utility. *Cumulative values are from the comparison of the HTAO algorithm against Q-routing. Percentage losses for both the system utility and energy results are from the end of the agent loss episode range of the* degraded *system.*

Algorithm	**Energy (average)**		**Energy (impact)**	**Energy (cumulative)**
	start	**finish**		
htao (stable)	81.5%	84.4%	n/a	600%
q-routing (stable)	81.5%	83.8%	n/a	0.0
htao (degraded)	77%	77%	−15%	230%
q-routing (degraded)	72%	72%	−20%	0.0

Table 12.5: WSN: Average and cumulative energy values. *Cumulative energy values are from the comparison of the HTAO algorithm against Q-routing. Percentage losses for energy are from the end of the agent loss episode range of the* degraded *system.*

Algorithm	Failed agents	Coverage
htao (degraded)	35%	78.5%
q-routing (degraded)	35%	73.0%

Table 12.6: WSN: Impact of loss of agents on coverage. *Impact on the average coverage in the* degraded *system when agents permanently fail over each subsequent episode.*

Algorithm	$\frac{\text{Quality}}{\text{energy}}$	Task-path depth	Energy distribution
htao (energy)	n/a	2.50	0.488
htao (quality)	11%	3.38	0.552
htao (distribution)	3%	3.06	0.572

Table 12.7: WSN: Effect of the composite task quality balance. *Results of varying the balance of the quality and energy components of the composite task quality equation and its impact on system utility optimisation and energy distribution.*

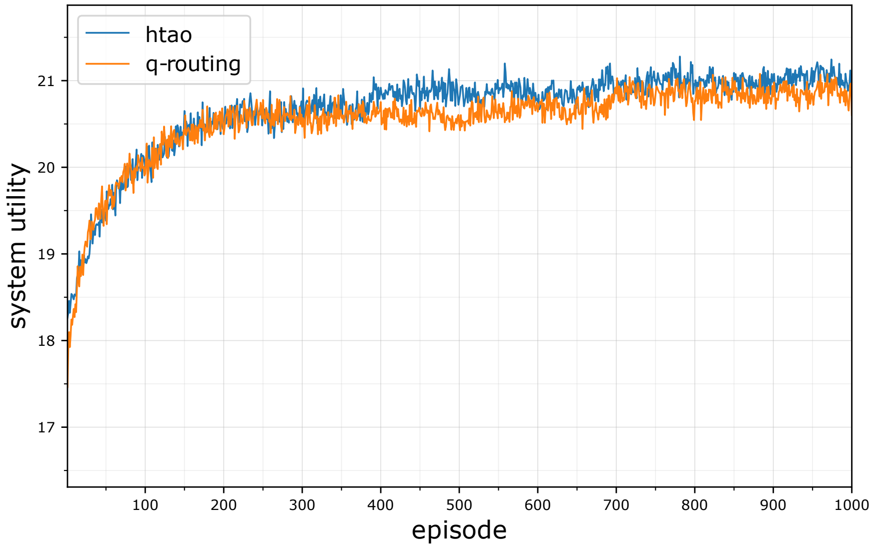


Figure 12.5: *Average system utility per-episode in the* stable *system.*

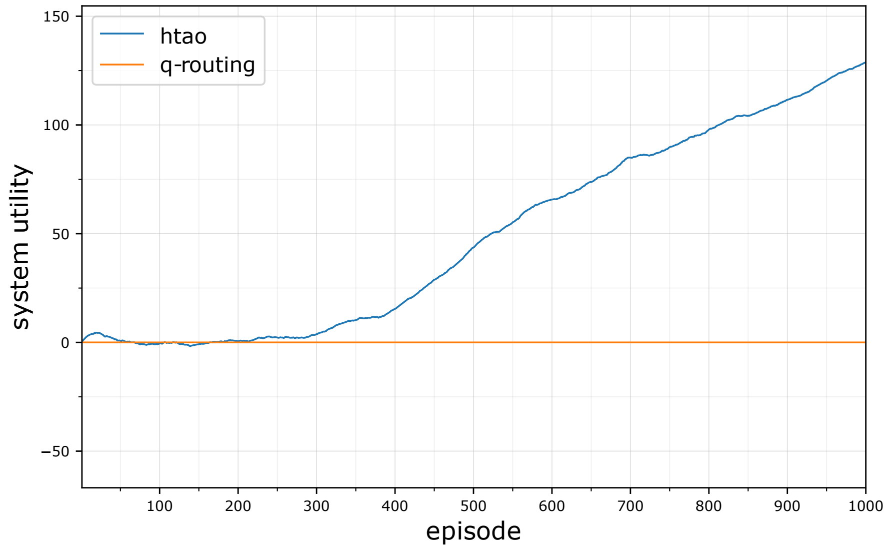


Figure 12.6: *Cumulative system utility in the* `stable` *system.*

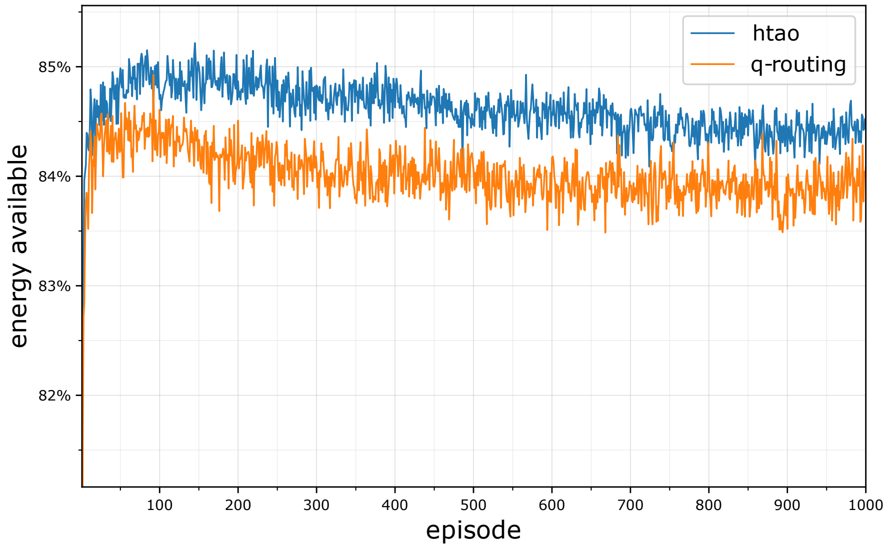


Figure 12.7: *Percentage energy available per-episode in the* `stable` *system.*

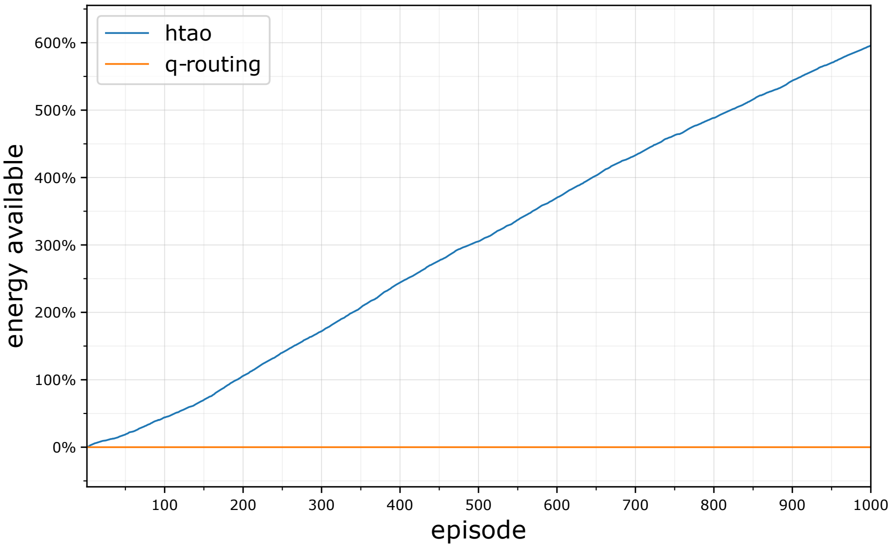


Figure 12.8: *Cumulative percentage energy available in the* `stable` *system.*

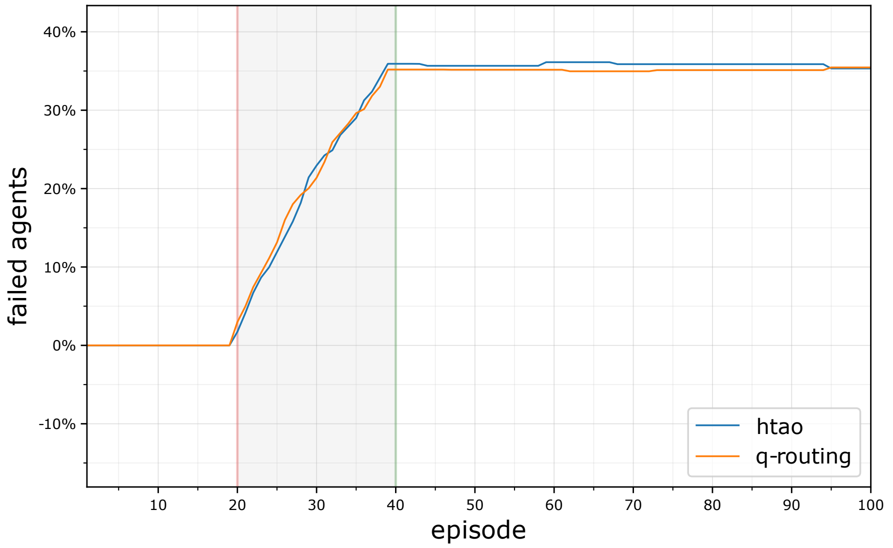


Figure 12.9: *Percentage of unavailable agents per-episode in the* `degraded` *system.*

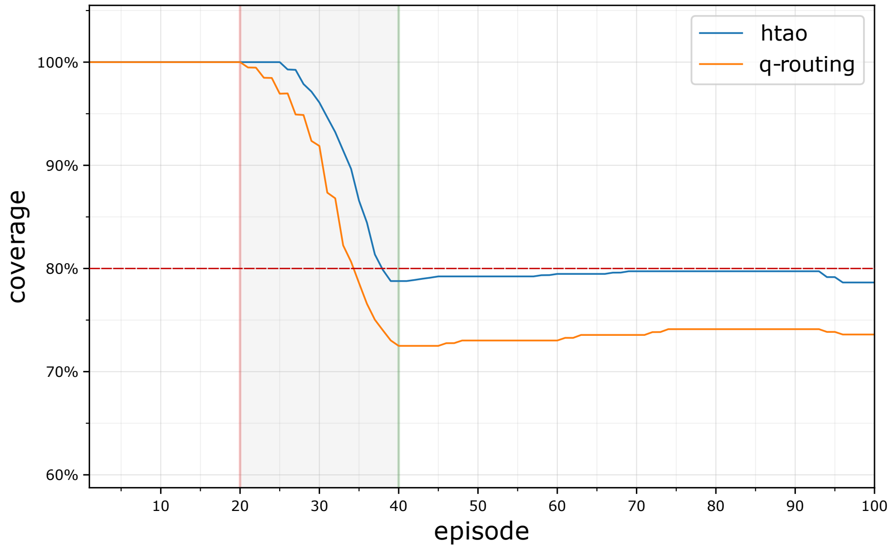


Figure 12.10: *System coverage in the* degraded *system.*

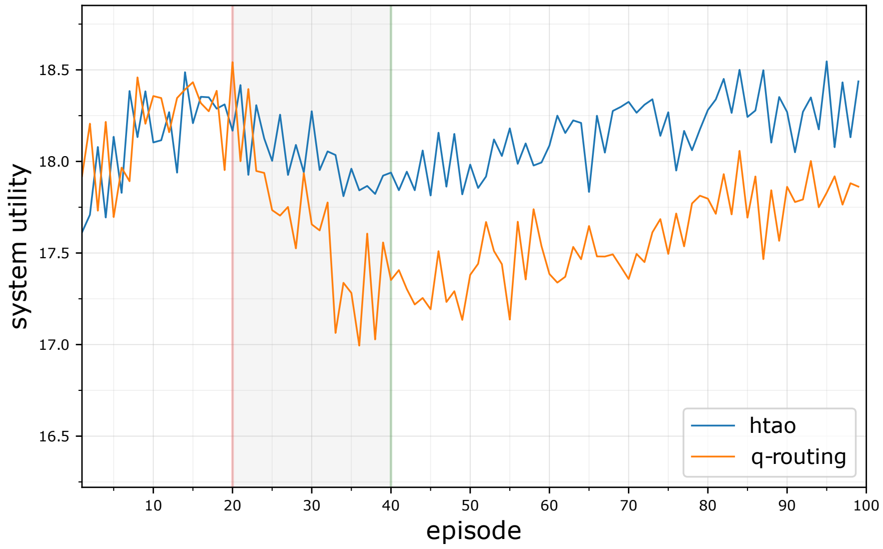


Figure 12.11: *System utility per-episode in the* degraded *system.*

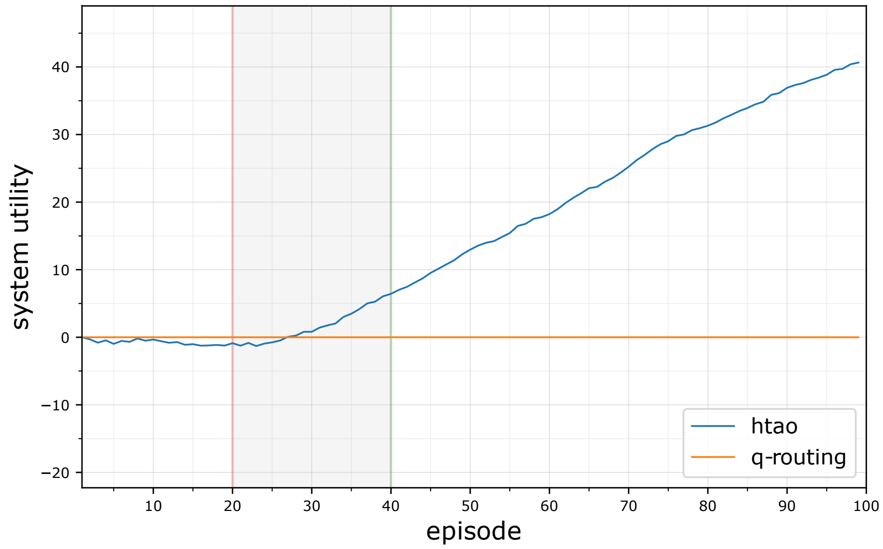


Figure 12.12: *Cumulative system utility in the* degraded *system.*

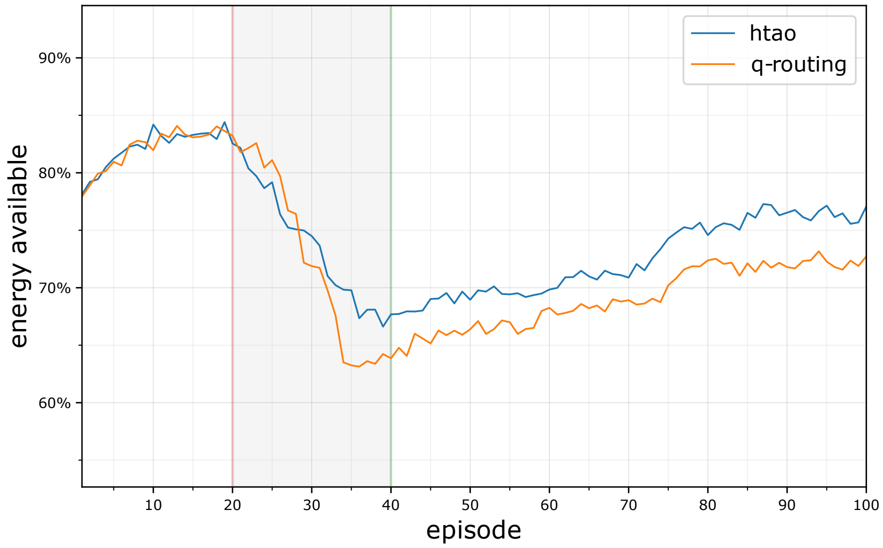


Figure 12.13: *Percentage energy available per-episode in the* degraded *system.*

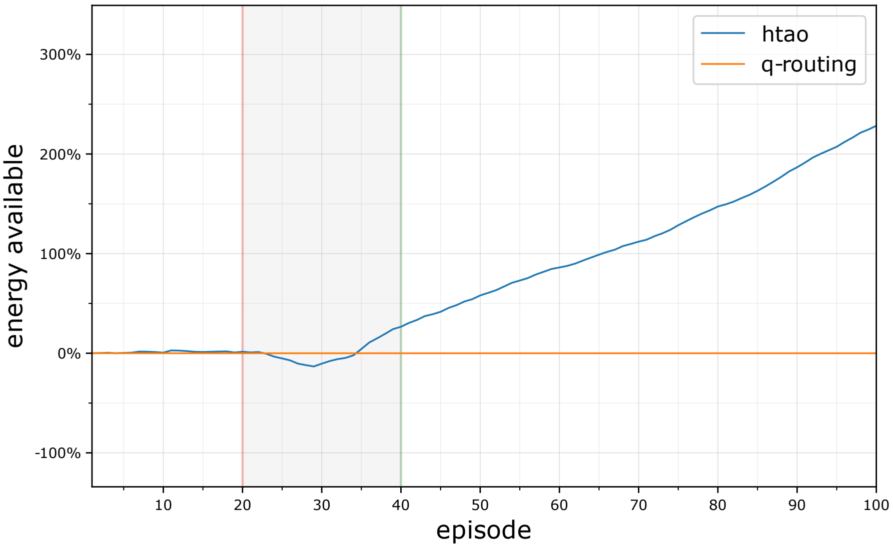


Figure 12.14: *Cumulative percentage energy available in the* degraded *system.*

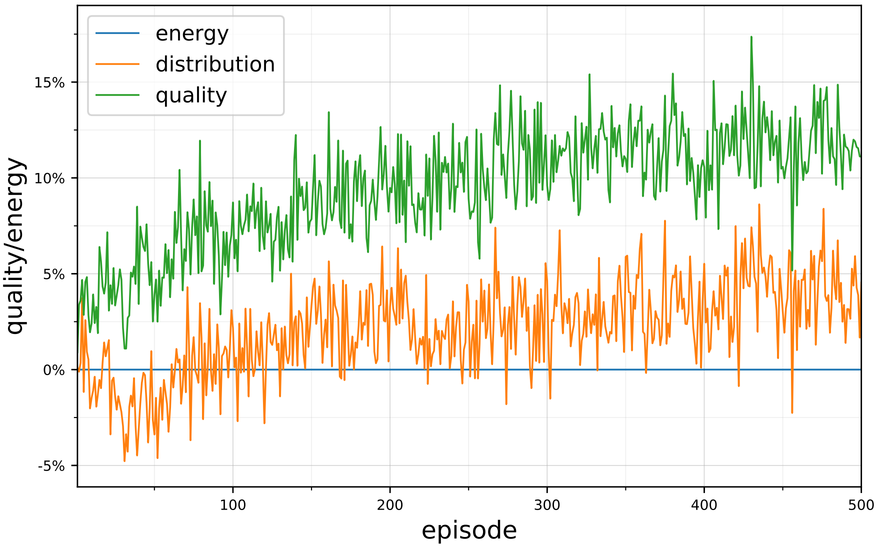


Figure 12.15: *Task quality to energy available ratio with htao (energy) as the baseline in the* biased *system*

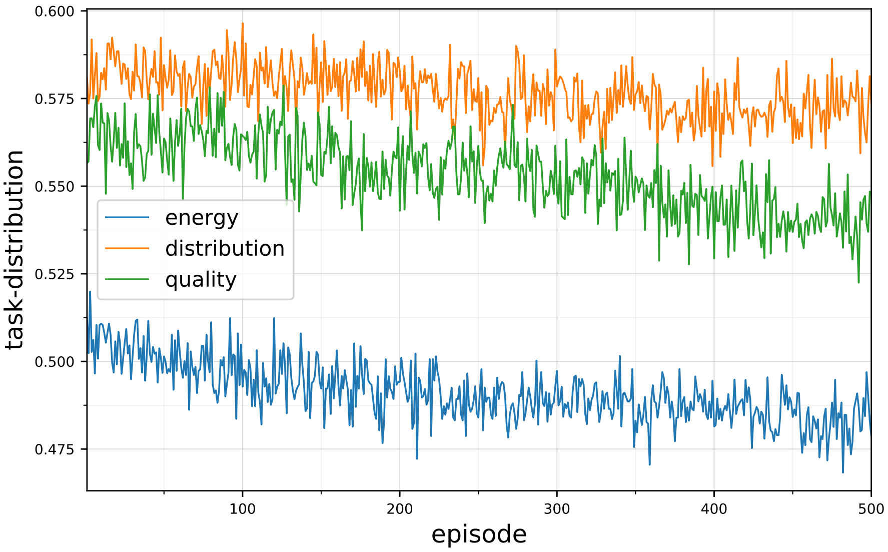


Figure 12.16: *The variance of unique sensor agents completing atomic tasks in the* `biased` *system.*

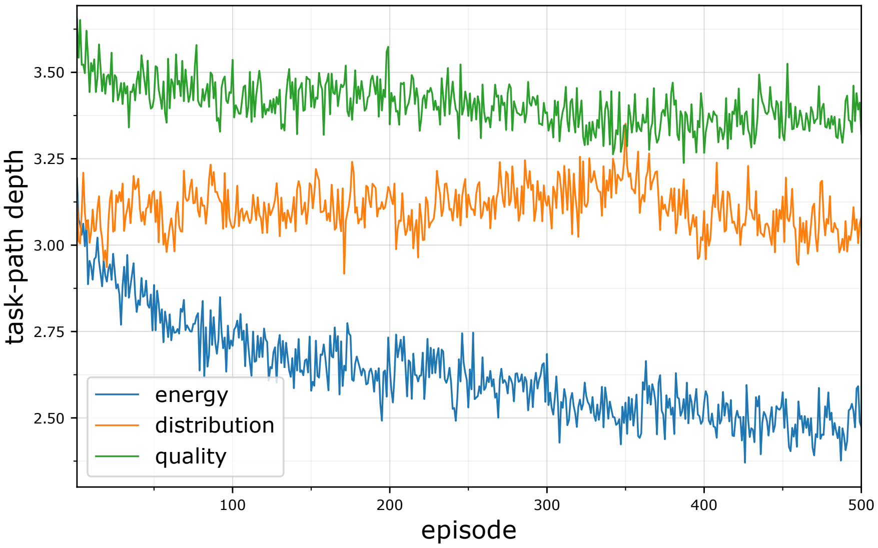


Figure 12.17: *Comparison of task-path depths in the* `biased` *system.*

Optimisation of utility, energy availability, and the impact of agent loss on coverage

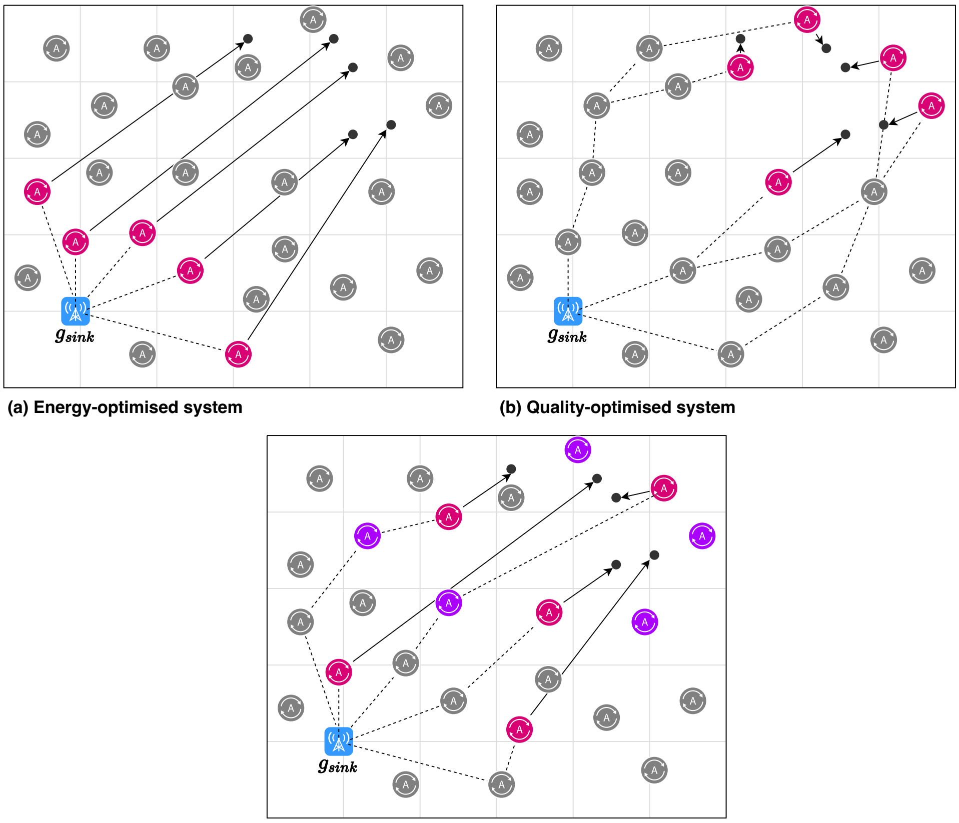
In the `stable` system, the HTAO algorithm optimised the system utility by 15.9% over 1000 episodes, compared to 15.4% for the q-routing algorithm (Figures 12.5 and 12.7). Energy availability also increased by 3.6% for htao algorithm and 2.8% for q-routing. Over the lifetime of the system the htao algorithm accumulated 138 more in utility than the q-routing comparison algorithm and 600% more energy availability (Figures 12.6 and 12.8).

From Figure 12.9 we can see that, in the `degraded` system, as we gradually simulate the loss of agents from episode 20 to 40, 35% of them become unavailable. The coverage of tasks using the htao algorithm falls by −21.5% during this period, a 5.5% improvement on the −27.0% drop of the q-routing algorithm. The ability of the system when utilising the htao algorithm to absorb agent availability loss better than the q-routing algorithm is also seen in both the impact on system utility, −2.1% as compared to −6.0%, and for the energy available, −2.1% compared to −6.0%.

Algorithm adaptability

We now look at the `biased` system in detail to examine how the algorithm varies the balance of optimisation over the composite task quality components, allowing multiple-objectives to be targeted. Figure 12.15 shows the task quality to energy availability ratio. As quality is optimised preferentially over energy availability, its values range higher. The htao (quality) algorithm configuration, with its higher c_{ql} value, optimises for atomic allocation quality and so these are increased by 13% over the htao (energy), and the htao (distribution) algorithm configurations by 3%.

In Figure 12.16 we see how completed atomic tasks are distributed in the system. This is measured by the variance in the individual agents that complete atomic tasks, where lower values indicate some agents are completing multiple tasks. In the htao (distribution) algorithm configuration, values remain relatively steady throughout the system lifetime at 0.579 to 0.572, dropping only −0.9%. The htao (quality) algorithm configuration starts slightly lower than this at 0.560 and falls 1.4% to 0.552. Notably the htao (energy) algorithm configuration starts and remains at a significantly lower distribution than both the other configurations at 0.488, dropping 3.4%. Figure 12.17 shows the related effect on task-path depths for each algorithm. htao (quality) and htao (distribution) have relatively stable task-path depths at 3.38 and 3.06 respectively. The average task-path depth of htao (energy) however drops from 3.02 to 2.50 over the system's lifetime, a 17.2% decrease.



(a) Energy-optimised system

(b) Quality-optimised system

(c) Distribution-optimised system

Figure 12.18: Three sample task-path patterns in the biased system. *In the htao (energy)-optimised configuration (a), the agents taking measurements are near the sink. Energy use is minimised but the atomic task-values are low. In the htao (quality)-optimised configuration (b) the agents taking measurements are close to the corresponding tasks' demand points, maximising their task-values. However, there is an increase in energy usage as they are more distant from the sink, with increased task-path depth. In the htao (distribution)-optimised configuration (c), the agents taking measurements are a mix of distances away from demand points, with the agents participating in the task-paths changing with different measurements.*

System behaviours

We use the illustrations in Figure 12.18 to give some intuition to our results. The results seen for htao (energy) relate most closely to the configuration in Figure 12.18(a), where energy consumption is reduced by using shorter task-paths and so, the amount of energy used by agents in the system to broadcast atomic task re-allocations is also decreased. The cost of this strategy is that the atomic tasks are completed to a lower quality. In Figure 12.18(c), task-path depths are large so that atomic tasks can be completed by agents close to the tasks' respective demand points, increasing atomic task quality as well as consumption, as seen in the results for htao (quality). The system in Figure 12.18(b) explains the results seen with the htao (distribution) algorithm configuration, where atomic tasks are completed by an increased number of different agents with different task-path depths and task qualities, giving a greater distribution of task completions and energy usage, but with more energy consumption overall than htao (energy), and less task-quality than htao (quality).

Summary of performance

Overall, the HTAO algorithm increased the utility of the realistic systems considered, reducing energy consumption while improving the quality of completed tasks. As compared to a multi-objective Q-routing algorithm, there was a 15.9% system utility improvement in the stable 25 node system over 1000 episodes, with energy availability increased by 3.6%. Evaluation of changing algorithm parameters to balance between energy availability, distribution, and task quality showed that these individual components could be prioritised in different ratios depending on the requirements of the optimisation required in the system. As seen in both the `stable` and `degraded` system configurations, utility and energy availability are both increased by the algorithm, and performance is improved as compared to the q-routing algorithm. Through the results of the `degraded` simulation we see that this performance extends to realistic system instability, adapting task routing to maintain coverage as agents are lost. Additionally, the results of the HTAO energy, quality, and distribution-optimised algorithm configurations in the `biased` system show that we can adapt the balance of the composite task value components through varying values of c_{pow}, c_{dst}, and c_{ql}, to achieve an adaptive multi-objective optimisation of the system. These results show that the algorithm optimises for the system goals for WSNs as laid out in Section 12.2, and is an improvement on the Q-routing approach used as a baseline.

Comparison to other algorithms

With a composite task value balance optimised solely to minimise energy consumption (i.e. $c_{pow} = 1, c_{dst} = 0, c_{ql} = 0$), the HTAO algorithm behaves similarly to energy-aware algorithms applied to WSN. In this configuration comparisons can be made with algorithms such as PEGASIS[347], or more closely to Q-routing algorithms like Q-probabilistic routing[348]. However, the task allocation and resource optimisation component of the HTAO algorithm is not accounted for in these implementations so provides only a comparison for the energy and route adaptation properties.

12.8 Summary

In this chapter the HTAO algorithm was evaluated on a model WSN system based on a realistic situation where agents would be randomly distributed across a geographical area, where maintenance and management would be challenging due to harsh or dangerous conditions. Our evaluation showed that the HTAO algorithm optimised for the composite task quality, energy available, and its distribution in the system, and that these components could be varied in their priorities through altering the domain-specific components of the allocation quality. This allowed the algorithm to balance across these different properties in the given systems and optimise for these multiple objectives in different ratios.

Part III

Applicability and analysis

Chapter 13

Conclusions and future work

In this final chapter we summarise our work and how this relates to our goals for this research. We also look at possible areas for future investigation.

13.1 Introduction

There are many systems that require the allocation of tasks amongst multiple agents in order for the system to meet its goals. These agents have dedicated access to their individual resources, which can be limited due to several factors; they possess fixed amounts of a resource that runs out, the bandwidth usage of the resource is constrained, or the resource is replenished at a fixed rate. Additionally, agents must form ad-hoc and adaptable communication networks amongst themselves to orchestrate and coordinate task completions.

When strategies for organising agents are planned centrally, the scale and complexity of the applicable systems is limited. However, with more decentralised approaches, the autonomy of agents means that the alignment of each agent's goals to that of the system is challenging. As agents use their own localised knowledge of the system for planning, they have a partial-view of the system, and therefore limited information on which they can base their choice of actions. As we move into real-world systems, we see an increased dynamism in the environment. With component and communication failures, as well as the loss and introduction of new agents to the system, agents cannot be relied on to be available to take part in tasks.

As we developed solutions that can be applied to systems with environmental dynamism and agent change, we chose wireless sensor networks (WSN) as our case study. These are large, distributed, agent-based systems found in a wide range of industries from vehicle-to-vehicle communications to large-scale environmental monitoring. Agents in these systems need to manage energy usage, maintain availability, distribute tasks effectively, and handle node communication failures. Decentralised algorithms are commonly used to meet these challenges, with hierarchical cluster formation or reinforcement learning techniques. There are challenges however in getting these algorithms to perform well where there are multiple objectives, agents are mobile, or connectivity varies over the lifetime of the system. These factors make them particularly useful to evaluate our work and its ability to meet our objectives for the research.

These are the motivations that led us to develop our original resource goals, objectives, and contributions (see Chapter 1, Sections 1.2, 1.4 and 1.5 respectively). In the next section, Section 13.2, we discuss the outcomes of our research as they relate to our contributions. We look at some possible applications of our work in Section 13.3, followed by future work on task and resource allocation, as well as improving our overall solution in Section 13.4.

13.2 Research and contributions

Contribution 1. Algorithms for task allocation

In Chapter 9 we developed the ATA-RIA algorithm as our first contribution (see Chapter 1, Contribution 1). This algorithm allowed each agent to improve their strategy for allocating tasks to other agents through the use of reinforcement learning. This algorithm used a number of sub-algorithms to achieve its performance, the RT-ARP, N-Prune, and SAS-KR algorithms. The RT-ARP algorithm used historical rewards trends as a form of intrinsic motivation to adapt the behaviour of agents, exploring the system for better agents to allocate tasks to when it was performing poorly, and focusing on task completion when it was performing well. In addition, the N-Prune and SAS-KR algorithms were used to manage the knowledge and neighbourhood of agents in order to ensure better quality retention of information while meeting the agent's constraints (e.g. such as CPU or memory limits).

Based on our evaluation in Chapter 9, Section 9.7, our solution performed to within 6.7% of the theoretical optimal with the system configurations considered. It provided 5× better performance recovery over no-knowledge retention approaches when system connectivity was disrupted. Overall the results demonstrated that the ATA-RIA algorithm met the desired objective of solving the task allocation problem in MAS.

Contribution 2. Algorithms for resource allocation

We introduced the MG-RAO algorithm in Chapter 10 as our second contribution (see Chapter 1, Contribution 2). This algorithm utilised reinforcement learning applied to estimates of the resource allocation demands of the task allocations coming from groups of agents. Each agent's estimates for multiple groups was then combined by using both the number of times different types of tasks had been allocated to it from each group, and the entropy of its resource allocation estimates.

The evaluation in Chapter 10, Section 10.5 was carried out in a simulated environment where multiple agents were allocated atomic tasks by other agents in the system. The MG-RAO algorithm showed a $23-28\%$ improvement over fixed resource allocation in the simulated environments. Results also showed that, in a volatile system, using the MG-RAO algorithm configured so that child agents model resource allocation for all agents as a whole had 46.5% of the performance of multiple group modelling. These results demonstrated the ability of the algorithm to solve resource allocation problems in multi-agent systems and to perform well in dynamic environments.

Contribution 3. Algorithms for hierarchical multi-objective task allocation

To meet our final contribution (see Chapter 1, Contribution 3), we combined our previous work from Chapters 9 and 10 to develop the HTAO algorithm of Chapter 11. This used the ATA-RIA algorithm for improving the allocation of tasks, and the MG-RAO algorithm for the allocation of an agent's limited resources to enhance the performance of atomic tasks with respect to groups of tasks with interdependent outcomes (see composite tasks, Section 8.2). Additionally, the HTAO algorithm added the ability for atomic tasks to be reallocated, so agents could form chains of task-relaying agents when finding the best way to complete an atomic task. Using this method, agents allocating tasks could extend their exploration of allocation policies across the system. This also allowed the development of more complex composite task quality functions that took into account system goals such as resource availability and resource usage distribution across the system, which we then went on to use for managing energy use in our WSN system in Chapter 12.

Application to a WSN system

Our case study was based on a realistic ocean-based WSN system, where not only was the quality of task completions an objective of the system, but where energy utilisation and distribution were also important. This tested our algorithms' ability to optimise toward multiple objectives. Overall, the HTAO algorithm increased the utility of the simulated systems considered, reducing energy consumption while improving the quality of completed tasks. As compared to a multi-objective Q-routing algorithm, there was a $\sim$ 15.9% system utility improvement in the stable 25 node system over 1000 episodes, with energy availability increased by $\sim$ 3.6%. Evaluation of changing algorithm parameters to balance between energy availability, distribution, and task quality showed that these individual components could be prioritised in different ratios depending on the requirements of the optimisation required in the system.

13.3 Applications

Application to real-world problems

The HTAO algorithm is applicable to a general class of problems where there are dynamic, self-organising networks, and where multiple agents need to learn to associate other agents with subtasks necessary for completion of a composite task. This work may be especially applicable to systems where there are changeable conditions that cause instabilities and where only limited maintenance or human intervention is possible. Future work would look at applying the HTAO algorithm to a larger scale, real-world system, testing how the algorithm performs in more complex environments.

There are examples of these in wireless sensor networks (WSN)[349], [350] where adaptive networking and optimisation are essential to keep usage and maintenance costs minimal. Similarly, there are uses in systems where agents are mobile such as vehicular ad-hoc networks (VANET)[114] and vehicle-to-vehicle communications (V2X) systems[4], [351]. Cloud computing service composition[352], [353] systems also provide real-world task allocation applications. As the HTAO algorithm is designed to work in dynamic environments, where optimisation targets are non-stationary, we expect that it will also be useful in these types of system.

We expect that testing practical deployment this work in the case of oceanographic or other environmental monitoring would be a productive next step[11]. The combination of harsh environmental conditions, difficulty of providing maintenance for remote agents, and mobility at slow speeds, should provide ideal conditions for successful use of HTAO.

13.4 Future work

Learning action impacts In Chapter 9, Section 9.4.3 we developed our understanding of how to measure the likelihood that taking a LINK action that changes an agent's neighbourhood composition, or an INFO action that changes its knowledge, will impact the choices of actions available to the agent in the future. This was based on the changes such actions made to neighbourhood and knowledge, and the probability that these changes would influence the performance of the agent when allocating tasks. However, in restricting the scope of this work we used estimated values for this impact for each type of action.

In future work we would expect to expand our research in this area to enable the agent to learn the impact of taking neighbourhood or knowledge-changing actions dynamically. Where an agent's actual experience of how its view of the system was altered, and how its performance changed on taking different types of actions would enable it to make more accurate predictions of the impact of doing so in the future. This should not only allow for an improved performance on the systems as evaluated in Chapter 9, but also enable the algorithm to be applied to systems with a greater number of differing task types, and where pre-calculating good values for their impact would be difficult.

Expansion of knowledge In Chapter 8, Section 8.2 we limited knowledge to the simple existence of other agents in the system. This meant that an agent that requested information from another agent in its neighbourhood gained a very limited amount of knowledge on doing so, the ability to link to new agents that were previously unknown to it. However, there is likely to be performance benefits achievable by expanding this knowledge to include other useful factors such as;

- *performance characteristics*, when an agent allocates tasks, it constructs a model through reinforcement learning of the ability of agents it allocates to to complete those tasks. However, this information is currently isolated to that agent, and eventually lost if it moves an agent out of its knowledge base. If we expand knowledge to include information on the learned performance of other agents, then this can be shared on request. If an agent that is allocating tasks then requests information, it then not only gains knowledge on the existence of other agents, but their relative abilities in performing specific task types, meaning the agent could improve its neighbourhood more quickly, compared to simply randomly searching the system for the best agents to allocate to;
- *resource availability*, we saw in Chapter 12 how agents could learn to minimise and balance energy use in a WSN system. This was done through adding domain-specific factors relating to that resource into the allocation quality of a set of atomic tasks, so as agents learned to allocate tasks, those factors were part of its understanding of the quality of the allocation of its tasks. Adding resource availability of neighbourhood agents to the knowledge shared on request would allow these agents to accelerate this learning and discover parts of the system with lower resource usage more quickly;
- *agent reliability*, component failure or degradation can make agents unreliable

in completing tasks they have been allocated. This unpredictability makes learning models for both task allocation, and resource allocation, more difficult for agents interacting with such an agent. Information an agent has accumulated on other agents about their reliability in completing its tasks for it, could be added to our definition of knowledge, and shared between agents when requested. This would enable agents to learn better models for both task and resource allocation, and more quickly.

Dynamic parent group assignment

In our work on parent groups of Chapter 10, Section 10.3.1, a child agent's parent groups were arbitrarily assigned. This rigid choice affects the ability of an agent to model each group's resource allocation needs well. There are a number of ways this approach to grouping could be improved, and made more adaptive to changes to the environment. For instance;

- *allocation stability grouping*, with parent-agent groups as currently specified, a parent agent with an unpredictable task allocation distribution would be assigned a parent-agent group randomly, which may contain agents with stable distributions. E.g. a parent-agent could be a faulty node in a network, making its task allocations unpredictable compared to fully working nodes. As these distributions are combined for modelling, the more random distribution reduces a child agent's ability to learn a useful model for that parent-agent group. To overcome this, parent agents can be grouped by a child agent by the variability in their atomic task allocations. By grouping together parent agents who reliably allocate them the same atomic tasks, the child agent can create a better model of the expected resource demands of that group compared to combining reliable and unreliable allocations in the same group;
- *predictive grouping*, a child agent can learn a model which predicts which parent agents should be grouped together to achieve the best allocation of its resources. Agents may have allocations of tasks that change in a predictable pattern, meaning a child agent can adapt its parent groups based on its prediction of these patterns. E.g. environmental monitoring agents that take more light sensor measurements during the day, or agents in a V2X system may allocate more tasks when they are non-stationary during predictable periods during the day;
- *similarity grouping*, there will be less change to the child agent's resource allocation model for a group of parent agents who have similar resource allocation requirements, than groups containing agents with very different demands. When parent agents' resource demands are very different, a child agent will alter its resource allocation model of that group significantly between one task allocation from a specific parent agent, and the next, due to the allocations from other parent agents in the intervening period. This problem could be significantly lessened by adapting parent agent groups composition as this similarity is learned.

Work to extend the arbitrary, fixed parent groups used in our work towards an adaptive algorithm that utilised some or all of the concepts above could be expected to achieve better performance in the resource allocation problem.

Part IV

Appendices

Appendix A

Mathematical notation

A.1 System and state

Symbol	Description
AT	a set of atomic tasks, typed by one of a set of atomic task types AP
CT	a set of composite tasks, typed by one of a set of composite task types CP
A	the set of actions that agents in the system can perform.
G	a set of agents, with each agent defined by a tuple $\langle id, c, r, \delta_k, \delta_n \rangle$
id	a unique identifier for an agent
$c \in AP$	an agent's capabilities
$r \in CP$	an agent's responsibilities
$\delta_n \in \mathbb{N}$	an agent's neighbourhood constraint, its communication constraint
$\delta_k \in \mathbb{N}$	an agent's knowledge, its memory constraint.
RES	a set of resources needed to perform tasks.

Table A.1: Summary of system notation.

Symbol	Description
$K \subseteq G$	the knowledge of an agent.
$N \subset K$	the neighbourhood of the agent.
G_S	the set of states of all agents in the system.
CL	the set of composite allocations in the system.
AL	the set of atomic allocations in the system.
ACT	the set of action allocations in the system.

continues on the next page...

...continued from previous page

RAL	the set of resource availabilities in the system.
RAL	the set of resource allocations in the system.
$\phi \in \mathbb{N}_0$	the current system counter.
$G_S(g)$	the agent state for an agent g.

Table A.2: Summary of state notation.

A.2 Actions

Equation	Description
$ASSIGN(e, ct)$	An *assign action* taken by an agent e, external to the system, to allocate a composite task ct.
$ALLOC(g, at, n)$	An *allocation action* taken by an agent g to allocate an atomic task at to an agent in its neighbourhood n.
$SINGLEALLOC(g, at, n)$	A *single allocation action* taken by an parent agent g to allocate an atomic task at to an agent in its neighbourhood n, which may not then be reallocated by that agent.
$EXEC(g, at)$	An *exec action* taken by an agent g to execute the atomic tasks at.
$INFO(g, n)$	An *info action* taken by an agent g to request information from an agent n in its neighbourhood.
$PROVIDE\_INFO(g, \widehat{g})$	A *provide information action* taken by an agent g to provide information to another agent epg.
$REMOVE\_INFO(g, k)$	A *remove information action* taken by an agent g to remove k from its knowledge.
$LINK(g, k)$	A *link action* taken by an agent g to add an agent from its knowledge to its neighbourhood.

Table A.3: Summary of actions.

Equation	Description
$concurrent(AL, g)$	The *concurrent atomic allocations* of an agent g are the atomic tasks currently allocated to that agent.
$atomics(AL)$	The *atomics* are the set of all atomic tasks included in an allocation.
$composites(CL)$	The set of all composite tasks included in an allocation.
$allocatable(CL, g)$	The atomic tasks available to allocate of an agent g as a result of a composite task allocation CL.

Table A.4: Summary of allocation selection equations.

Equation	Description
$complete(AT)$	The *atomic task completion* function returns 1 if all of the atomic tasks in the given set AT have been executed by agents, and the results returned to the agent whose composite task contains the atomic task, and 0 otherwise.
$wait(g)$	The *agent wait* function sets an agent g to wait for the system time counter to move one step, $\phi \rightarrow \phi + 1$.

Table A.5: Summary of orchestration equations.

Equation	Description
$category(a)$	The *action category* of an action a, its action type.
$actions(g, s)$	An agent g's *available actions*, all the actions that the agent g can perform in its current state.
$targets(g, s, G)$	An agent g's *target actions*, all the available actions for the agent when in a state s, that interact with an agent in the set G.
$unknown(g, S)$	An agent g's *unknown states*, states an agent could be in, but lacks the knowledge to access.

Table A.6: Summary of action equations.

A.3 Task allocation

Equation	Description
$permutations(AT, G)$	The *task permutations*, the possible different ways of allocating a set of atomic tasks AT amongst a set of agents G.
$allocql(AT, AL)$	The *allocation quality* of a set of atomic tasks AT is the combined atomic task quality of their completions.
$utility(S)$	The *utility* of the system over a set of states S.
$utility^*(S)$	The *theoretical optimal utility* of the system over a set of states S.

Table A.7: Summary of task quality equations.

Equation	Description
$optallocql(AT, G, AL)$	The *locally-optimal allocation* of tasks T to set of agents G is the allocation that gives the maximum quality.
$allhoods(g, G)$	Returns all the possible neighbourhoods of an agent g given a set of agents G.

continues on the next page...

...continued from previous page

$opthood(AT, g, G, AL)$	The *optimal neighbourhood* returns the neighbourhood in a set of agents G of the an agent g that gives the best quality given a set of atomic tasks AT and an allocation AL.
$sysalloc(AT, g, G, AL)$	The *system-optimal allocation* is the locally-optimal allocation to the optimal neighbourhood.
$sysallocql(AT, g, G, AL)$	The *system-optimal quality* is the quality of the system-optimal allocation.
$jointallocql(AT, G)$	The *joint-optimal allocation* is the maximum theoretical quality of allocation of atomic tasks AT amongst agent G.

Table A.8: Summary of task allocation equations.

Equation	Description
$rlupdate(g, oldstate, newstate, a, reward, Q)$	The *reinforcement learning function* updates the Q-table values of an agent g as it transitions from state s_t to s_{t+1} through taking action a for which it received reward r.

Table A.9: Summary of reinforcement learning equations.

Equation	Description
$samples(SP, A)$	The *action sample selection* function returns the action samples of a set of actions A.
$actval(SP, A, t)$	The *action information quality* function is a proxy for the value of information collected about an action a up to a time t, given the set of action samples SP.
$hoodval(SP, g, G)$	The *neighbour information value* as the sum of the quality values of all action samples SP of an agent g that refer to actions that involve agents in G.
$minhoodval(SP, g)$	The *minimum value neighbour* of an agent g is the agent that generates the least neighbour information value.

Table A.10: Summary of action value equations.

Equation	Description
$hoodimpact(AT, N^{'}, N^{''}, AL)$	The *neighbourhood impact* is the difference between the locally-optimal allocation qualities between neighbourhoods $N^{'}$ and $N^{''}$ given a set of atomic tasks AT and allocation AL.

continues on the next page...

...continued from previous page

$maxhoodimpact(AT, G, AL)$	The *maximum neighbourhood impact* is the maximum possible neighbourhood impact given a set of atomic tasks AT and all combinations of neighbourhoods that can be formed from a set of agents G.
$knowimpact(AT, K^{'}, K^{''}, AL)$	The *knowledge impact* of an agent changing its knowledge from set of knowledge $K^{'}$ to $K^{''}$ is the difference between the maximal neighbourhood impacts of the respective sets of agents in each knowledge base.
$actimpact(AT, N^{'}, N^{''}, K^{'}, K^{''}, AL)$	The *action impact* is the expected value of the change in possible allocation quality if an action a is taken.

Table A.11: Summary of action impact equations.

Equation	**Description**
$localdist(AT, g, AL)$	The *locally-optimal allocation metric* is the difference between an agent's current allocation quality of atomic tasks AT to agents in its neighbourhood, and the locally-optimal allocation quality.
$systemdist(AT, g, G, AL)$	The *system-optimal allocation metric* is the difference between an agent's current allocation quality and the system-optimal allocation quality given the set of agents in the system, G.
$positiveact(a, AT, g, G, AL)$	The likelihood that an agent g allocating a set of atomic tasks AT in a system of agents G with allocation AL will improve its performance by taking an action a.
$impactinterpol(x)$	The *impact interpolation function* is generated by taking a linear interpolation over the rows of a TSQM.
$impacttrans(\mathtt{w})$	The *impact transformation function* estimates the probability that taking an action from an action-category in the current neighbourhood will be positive .

Table A.12: Summary of action metrics equations.

A.4 Resource allocation notation

Equation	Description
$resallocation(g, ap, RAL)$	An agent's *resources* mapping is the amount of resources allocated to that agent for a specific task type.
$allocql(AT, AL, RAL)$	The *allocation quality* of a set of atomic tasks AT is the combined atomic task quality of those atomic tasks' completions, given the resource allocation RAL.
$domtask(at, g)$	The *domain-specific atomic quality* component, scales the atomic task quality, specific to a particular application.
$domql(at, g)$	The *domain-specific allocation quality* component, captures additional characteristics of allocation quality specific to a particular application.
$propval(at, ct)$	The *proportional value* of a component atomic task of a composite task ct is the fractional value each atomic task contributed to ct as judged after ct's completion.
$compositeql(CT, AL, RAL)$	The *composite task quality* is the quality given by a set of composite task CT performed under atomic task allocation AL and resource allocations RAL.
$absval(ct, at, AL, RAL)$	Component tasks absolute value, The absolute value of each component atomic task at of a composite task ct.

Table A.13: Summary of resource task value equations.

Equation	Description
$group(g, G)$	A *parent group* is defined as a fixed mapping for each child agent to sets of parent agents that are grouped together for task value modelling.
$weight(cg, G, ap, rp)$	The *parent group task weights*, for each parent group of an agent g, there is a set of weights for each of its resources res.
$\mathbf{PW}_{rp}$	The *parent group task weights matrix* , the parent group task weights for all parent groups of an agent g and resource res.

Table A.14: Summary of resource group equations.

Equation	Description
$allocnum(g, G)$	The *parent agent sample count* is a mapping of a set of agents G to a count of tasks allocated to an agent g by a member of that group.
$kullback(g, G, AP, rp)$	The *parent group weights entropy* is the relative entropy of the resource weights of a parent group G of an agent g.
$BL = blend(g, G, AP, rp)$	The *resource weights blending vector* is a vector that combines sample counts and resource weight entropy for each parent-group.
$combined(BL, \mathbf{PW}, rp)$	The *combined resource weights function* takes the parent group weights matrix $\mathbf{PW}_{rp}$ and the blending matrix BL as parameters and outputs a vector of resource weights for atomic task types.
$index(pg, ap, rp)$	Parent-task type index mapping, utility function to map a specific parent agent and atomic task type to an index in a child agent's parent agent weights matrix.
$\mathbf{E}$	Eligibility trace matrix, a matrix of values to be used to map absolute task values to multiple past actions.
$updatetrace(\mathbf{E}, pg, ap, rp, \gamma')$	The *eligibility trace update* algorithm used to update the eligibility trace matrix.

Table A.15: Summary of resource allocation equations.

A.5 Hierarchical allocation notation

Symbol	Description
$sink(at)$	A *sink* of a task at from a set of agents G is an agent that receives composite tasks from outside the system.
$relays(at)$	A group of *relay*s of a task at from a set of agents G is the sequence of agents that are allocated the atomic task, but do not complete it, instead re-allocating it to other agents.
$executor(at)$	An *executor* of a task at from a set of agents G is the unique agent that executes a given atomic task.
$path(at)$	A *task-path* is a mapping of an atomic task at to the ordered sequence of agents from a set G that work together to complete that task.
$pathcost(at, G, rp)$	*task path cost* is the resources that must be used by agents to participate in a task path.

Table A.16: Summary of hierarchical allocation equations.

A.6 WSN notation

Symbol	Description
$deploy(g)$	An agent's *deployment* is its mapping to a specific (x, y) location in a coordinate system.
$demand(at)$	An atomic tasks' *demand point* is the specific location targeted by the task, i.e. a location to take a sensor measurement.
$range(at, g)$	The distance between an agent g's deployment location, and the demand point of an atomic task at.
$resaverage(G, rp, RAV)$	The average resource availability of a resource type rp across a set of agents G, given resource availability RAV.
$resdist(G, rp, RAV)$	The *resource distribution* is the inverse variance of the availability of a resource type rp across a group of agents G, given resource availability RAV.

Table A.17: Summary of WSN system equations.

Symbol	Description
$success(ct, at)$	The *atomic task success* of an atomic task at to a composite task ct is a function that maps to 0 if the atomic tasks' result is not considered of high enough quality to be useful to the composite task, 1 otherwise.
$syscov(CT)$	The *system coverage* is the fraction of those atomic tasks composing the composite tasks which were successful.
$resilience(CT, G', G)$	A systems' *task resilience* measures the change in system coverage as the number of agents in the system change.

Table A.18: Summary of WSN metrics equations.

Appendix B

Extended system concepts

B.1 State-action space size

Calculating the size of states

The actions possible for an agent g to take in a state are defined by;

1. the set of types of the atomic tasks it still has to allocate, AP;
2. the agent's current neighbourhood N, which dictates the number of $ALLOC$ and $INFO$ actions an agent can take;
3. agent's current knowledge K, which gives the number of $LINK$ actions.
4. the number of actions types in AP it has in its capabilities $c(g)$, giving $EXEC$ actions.

Therefore the state size for an agent g can be given by a function $statesize\colon \mathcal{G}\times 2^{\mathcal{AP}} \times 2^{\mathcal{G}} \times 2^{\mathcal{G}} \longrightarrow \mathbb{R}_{>0}$:

$$\begin{aligned} statesize(g, AP, N, K) &= \substack{\text{Number of}\\ \text{EXEC actions}} + \substack{\text{Number of}\\ \text{ALLOC actions}} + \substack{\text{Number of}\\ \text{INFO actions}} + \substack{\text{Number of}\\ \text{LINK actions}} \\ &= |c(g) \cap AP| + |N||AP| + |N| + |K| \\ &= |c(g) \cap AP| + |N|(|AP| + 1) + |K| \end{aligned} \tag{B.1}$$

Given an agent g has responsibilities $r(g)$, permutations of a set of unique atomic task types AP, then the maximum state-space size possible is, $maxspacesize\colon \mathcal{G} \times 2^{\mathcal{AP}} \times 2^{\mathcal{G}} \times 2^{\mathcal{G}} \longrightarrow \mathbb{R}_{>0}$, where:

$$maxspacesize(g, AP, N, K) = \sum_{\forall AP' \subseteq AP} statesize(g, AP^{'}, N, K) \tag{B.2}$$

Size of Q-tables

As neighbourhoods and knowledge are dynamic, and agents retain the Q-values of past states, then, without pruning, the possible Q-table size of an agent with full system knowledge would be $spacesize(g, AP, G, G)$. However, we use pruning strategies such as SAS-KR to maintain a reduced Q-table by removing Q-values that correspond to states and actions involving agents no longer in an agent's knowledge when they judged to be less useful to improving performance through SAS-KR (See Chapter 9, Sections 9.4.2). We illustrate this through Figure B.1, which shows the size of agents'

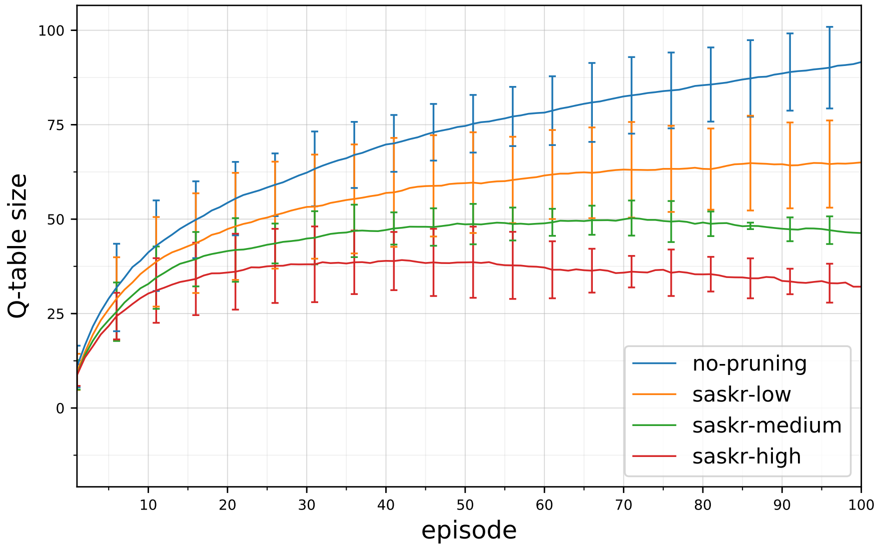


Figure B.1: The average Q-table sizes. *Average Q-table sizes of agents for the ATA-RIA algorithm in the* `stable` *system of Chapter 9, Section 9.6. Where SAS-KR is configured using no-pruning, low, medium, and high $\hat{\mu}_{\mathtt{min}}$ values.*

Q-tables over the course of 100 episodes, with no-pruning, and low, medium, and high values for $\hat{\mu}_{\mathtt{min}}$ for the SAS-KR algorithm, run in the `stable` system in Chapter 9, Section 9.6. Data and p-values are shown in Table B.1 for completeness.

Algorithm	$\hat{\mu}_{\mathtt{min}}$	**Q-table size**		**p-value**
		(max)	(100 episodes)	
no-pruning	n/a	91.6	91.6	0.98
saskr-low	0.01	65.0	65.0	0.62
saskr-medium	0.05	50.3	46.3	0.99
saskr-high	0.10	39.2	32.1	0.99

Table B.1: Q-table sizes for the stable system in Chapter 9, Section 9.6 . *The table shows the maximum and final (100 episodes) average Q-table sizes of agents for the ATA-RIA algorithm in the* `stable` *system of Chapter 9, Section 9.6. Where SAS-KR is configured using no-pruning, low, medium, and high $\hat{\mu}_{\mathtt{min}}$ values. P-values are T-tests for the null hypothesis that the utilities in the final episode are equal to the population mean with significance level $\alpha = 0.05$.*

B.2 The likelihood of selecting action types

For each state s, the proportion of task allocation, neighbourhood changing, and knowledge changing actions is given by:

$$\begin{aligned}&\left(\begin{array}{c}\text{Number of}\\\text{EXEC actions}\end{array}\times\begin{array}{c}\text{Number of}\\\text{ALLOC actions}\end{array}\right):\begin{array}{c}\text{Number of}\\\text{LINK actions}\end{array}:\begin{array}{c}\text{Number of}\\\text{INFO actions}\end{array}\\&=\big(|c(g)\cap AP|+|N||AP|\big):|K|:|N|\end{aligned}\tag{B.3}$$

Example B.2.1 (*Example of probabilities of action selection*). A system with $|cp| = 10$, $|N| = 10$, $|K| = 20$, with random distribution action selection, the ratio of action selections will be $100 : 20 : 10$ (for an agent with no EXEC actions, i.e. where $c(g) = \emptyset$). The probabilities of taking these action types will therefore be:

$$\begin{aligned}P(EXEC + ALLOC) &= 0.77\\P(LINK) &= 0.15\\P(INFO) &= 0.08\end{aligned}\tag{B.4}$$

In other words, there is approximately a 25% chance an agent will select an action that will change its neighbourhood or knowledge, and may lose learned Q-values as a result of the action.

Impact of action sampling

With no differentiation between action types, algorithms will sample the neighbourhood and knowledge-changing actions equally with task allocation actions. Not only does this mean that an agent will take longer to learn allocations for the atomic tasks they have, but also, with each neighbourhood changing action they sample, they change the states that are accessible to them, and need to learn over new actions. Additionally, with each sampling of a knowledge changing action, they may lose learned information on the actions related to that knowledge that would lead to better task allocation choices due to constraints on memory.

This is one reason why algorithms that do not distinguish between action types in sampling see increased times to complete each episode compared to our algorithms. In smaller systems there is a reasonable likelihood that the agent may have already learned some information about the states introduced by the neighbourhood agent or knowledge agent introduced. As the size of the system increases, and $|K| \lll |G|$, each agent learned about is likely to be new to the agent's knowledge, meaning that value of its action choices must be learned from scratch.

The combination of 2 factors affects the ability of some algorithms to function well in our simulation systems; not distinguishing between action-types, leading to neighbourhood and knowledge changes, and subsequently the loss of learned Q-values; and that our systems had background perturbations, i.e. the temporary loss of availability of agents, exacerbating the problems of learning action-selections. These factors greatly increase the likelihood of choosing actions that do not move the agent's task closer to completion when using the `<qlboltz>` and `<qlreset>` comparison algorithms in the simulations in Chapter 9, Section 9.6.

Appendix C

Task allocation

C.1 Parameters for system simulations and algorithms

Variable	Summary	Value
$\lvert AP\rvert$	Number of atomic task types	20
$\lvert CP\rvert$	Number of composite task types	10
$\lvert K(g)\rvert$	Size of an agent's knowledge	7
$\lvert N(g)\rvert$	Size of an agent's neighbourhoods	5
$\lvert ap \in cp\rvert$	Number of atomic tasks composing a composite task type	5
n/a	Frequency distribution of composite tasks' arrival in the system	One *cp* per parent agent per episode
ω_g	The atomic task quality produced by a child agent for a task.	$(0, 1]$

Table C.1: General parameter values.

Variable	Summary	Optimal	Exploration	Volatile	Large
$\lvert PG\rvert$	Number of parent agents in the system	3	3	3	10
$\lvert CG\rvert$	Number of child agent in the system	10	10	10	$\{10, 50, 100\}$
W	The approximate action-impact values	$\{(\text{LINK}, 0.10), (\text{INFO}, 0.20)\}$	$\{(\text{LINK}, 0.10), (\text{INFO}, 0.20)\}$	$\{(\text{LINK}, 0.10), (\text{INFO}, 0.20)\}$	$\{(\text{LINK}, 0.10), (\text{INFO}, 0.20)\}$ $\{(\text{LINK}, 0.10), (\text{INFO}, 0.55)\}$ $\{(\text{LINK}, 0.10), (\text{INFO}, 0.60)\}$

continues on the next page...

...continued from previous page

$P(leave/join\|pg)$	Probability of agent leaving or re-joining the system each episode	0	0	0.01	0

Table C.2: Simulation parameter values.

C.2 Summary of results

Algorithm	% performance decrease from <optimal> (best)
<ataria>	6.7%
<qlreset>	181.0%
<qlboltz>	306.6%(235.0%)

Table C.3: Experimental results for the stable system after 100 episodes.

Algorithm	% performance increase over <rtrap^0>
<rtrap$^-$>	44.3%
<rtrap$^+$>	67.0%

Table C.4: Experimental results for the exploration system after 100 episodes.

Algorithm	% performance decrease from <nodrop>
<drop>	9.7%
<nosaskr>	54.6%

Table C.5: Experimental results for volatile system after 100 episodes.

Algorithm	% performance decrease from <large-optimal>
<large-25>	3.6%
<large-50>	7.2%
<large-100>	8.6%

Table C.6: Experimental results for large system after 100 episodes.

Statistic	<optimal>	<rtrap^0>	<nodrop>	<large-optimal>
mean	53.75	75.91	108.46	28.36
std	0.54	8.98	22.85	6.55
min	52.55	67.75	70.30	24.93
25%	53.38	70.31	89.47	25.63
50%	53.80	72.12	111.61	25.98
75%	54.12	77.76	124.32	27.72
max	55.01	113.94	214.75	72.81

Table C.7: Statistics of baseline algorithms results.

Label	p-value	Label	p-value	Label	p-value
<optimal>	0.54	<nodrop>	0.86	<large-optimal>	0.76
<ataria>	0.87	<drop>	0.87	<large-25>	0.67
<rtrap^0>	0.33	<nosaskr>	0.47	<large-50>	0.87
<rtrap$^-$>	0.29			<large-100>	0.91
<rtrap$^+$>	0.50				

Table C.8: Final episode p-values of algorithm results. *T-test for the null hypothesis that the utilities in the final episode are equal to the population mean with significance level* $\alpha = 0.05$.

C.3 Calculating approximate action-impact values

We ignore actions apart from $INFO$ and $LINK$ as these are the only ones that alter the neighbourhood, N, or knowledge, K, of an agent. To make our first approximations we assume that the selection of actions of these two types is distributed uniformly. Given this, the probabilities of changing the neighbourhood or knowledge for these action types is:

$$\text{LINK: } P(N^{'}, N^{''}) = \frac{1}{2} \times \frac{|N|}{|K|}, \text{ and } P(K^{'}, K^{''}) = 0 \tag{C.1}$$

$$\text{INFO: } P(N^{'}, N^{''}) = 0 \text{ and } P(K^{'}, K^{''}) = 1 - P(N^{'}, N^{''}) \tag{C.2}$$

For $LINK$ actions, an agent will be replaced in N with one of the agents in K. We make the simplification that the neighbourhood impact of the actions will be 1 if we add an agent that can complete a task type better than an existing agent in the neighbourhood, and 0 otherwise. Assuming agents are distributed randomly across $N \cup K$, on average a $LINK$ action will produce a neighbourhood impact of $1 - \frac{N}{K}$. Therefore, we get an approximation of action impact of:

$$\left(\frac{1}{2}\frac{|N|}{|K|}\right)\left(1 - \frac{|N|}{|K|}\right) \tag{C.3}$$

For $INFO$ actions an agent in K will be replaced by one in G. Again, assuming agents are distributed randomly across $K \cup G$, on average an $INFO$ action will produce a knowledge impact of $1 - \frac{|K|}{|G|}$. Therefore:

$$\left(1 - \frac{1}{2}\frac{|N|}{|K|}\right)\left(1 - \frac{|K|}{|G|}\right) \tag{C.4}$$

So that:

$$W = \left\{\left(\text{LINK}, \left[\frac{1}{2}\frac{|N|}{|K|}\right]\left[1 - \frac{|N|}{|K|}\right]\right), \left(\text{INFO}, \left[1 - \frac{1}{2}\frac{|N|}{|K|}\right]\left[1 - \frac{|K|}{|G|}\right]\right)\right\} \tag{C.5}$$

Example C.3.1 (*Optimal allocations in multi-agent systems*). A system contains 100 agents, with each agent's neighbourhood size being 10, and its knowledge 20. Using the above approximation we get action-impact values:

$$W = \left\{\left(\text{LINK}, \frac{1}{8}\right), \left(\text{INFO}, \frac{3}{5}\right)\right\} \tag{C.6}$$

Appendix D

Resource allocation

D.1 Parameters for system simulations and algorithms

Variable	Summary	Value
$\lvert AP\rvert$	Number of atomic task types	20
$\lvert CP\rvert$	Number of composite task types	10
$\lvert ap \in cp\rvert$	Number of atomic tasks composing a composite task type	5
tf	Frequency distribution of composite tasks' arrival in the system	One cp per pg per episode
$\lvert RP\rvert$	Number of resource types needed to perform tasks	1
q_{cg}	The atomic task quality produced by a child agent for a task.	$(0, 1]$
ctv	The component tasks value produced by a parent agent for atomic tasks that are part of its composite task.	$(0, 1]$

Table D.1: General parameter values.

Variable	Summary	Single	Multi	Volatile	Large
$\lvert PG\rvert$	Number of parent agents in the system	10	10	10	50
$\lvert CG\rvert$	Number of child agent in the system	1	3	1	1
$\lvert group(g, G)\rvert$	Parent agent group size	1	1	1	$\{1, 2, 5, 10, 25, 50\}$

continues on the next page...

...continued from previous page

$P(leave/join \mid pg)$	Probability of parent-agent leaving or re-joining system per-episode	0	0	0.25	0

Table D.2: Simulation parameter values.

D.2 Summary of results

Label	p-value
`<mgrao-max>`	0.899
`<mgrao-1:1>`	0.993
`<uniform>`	0.977

Table D.3: Final episode p-values of algorithm results. *T-test for the null hypothesis that the utilities in the final episode are equal to the population mean with significance level* $\alpha = 0.05$.

Appendix E

Case study

E.1 The multi-objective Q-routing algorithm

The variation of Q-routing we use for comparison utilises the same multi-objective reward function as HTAO. It also assumes full knowledge and neighbourhood containing all agents in the system. The algorithm uses the update function for actions to Q-value mappings from the ATA-RIA algorithm. In comparison to the ATA-RIA algorithm, it uses ϵ-greedy selection for choosing actions instead of RT-ARP, and does not attempt to learn actions relating to knowledge and neighbourhood as agents assume full system information. This algorithm follows strategies developed from Q-routing[345], but without low-level details on node communications and protocols. This is comparable in particular to work that adds energy-awareness and adaptive routing[354]–[356]. We also adapt the application of Q-learning to fit the multi-agent, task-based framework.

ALGORITHM 10: The multi-objective Q-routing algorithm

Input: g , the agent allocated the composite task.
Input: ct , the composite task allocated to the agent.
Input: Q , the Q-table of agent g.
Output: $Q^{'}$, the updated Q-table of agent g.

```
for $at \in ct$ do
    // Store current system state
    $oldstate \leftarrow s$
    $a \leftarrow \epsilon$-greedy($Q[s, A]$)
    if $a = EXEC(g, at)$ then
        $EXEC(g, at)$
        if $at$ is successfully completed then
            $ct \leftarrow ct \setminus \{at\}$
        end
    else if $a = ALLOC(g, at, n)$ then
        $ALLOC(g, at, n)$
        if $at$ is successfully completed then
            $ct \leftarrow ct \setminus \{at\}$
        end
    // Store new system state
    $newstate \leftarrow s$
    // Update Q-value mappings using reward generated by action
    $Q^{'} \leftarrow rlupdate(oldstate, newstate, a, reward(a), Q)$
end
return $Q^{'}$
```

E.2 Summary of results

Label	p-value
htao (stable)	0.517
q-routing (stable)	0.436
htao (energy)	0.869
htao (quality)	0.556
htao (distribution)	0.442

Table E.1: Final episode p-values of algorithm results. *T-test for the null hypothesis that the utilities in the final episode are equal to the population mean with significance level* $\alpha = 0.05$.

List of Figures

List of Tables

Alphabetical Index

Bibliography

[1] F. Derakhshan and S. Yousefi, “A review on the applications of multiagent systems in wireless sensor networks,” *International Journal of Distributed Sensor Networks*, vol. 15, no. 5, May 2019. DOI: 10.1177/1550147719850767. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1550147719850767.

[2] J. Early. “What’s hot in Multi-Agent Systems.” (2020), [Online]. Available: https://medium.com/swlh/whats-hot-in-multi-agent-systems-4b0f348e68bd.

[3] I. Althamary, C. W. Huang, and P. Lin, “A survey on multi-agent reinforcement learning methods for vehicular networks,” *2019 15th International Wireless Communications and Mobile Computing Conference, IWCMC 2019*, pp. 1154–1159, Jun. 2019. DOI: 10.1109/IWCMC.2019.8766739.

[4] W. Tong, A. Hussain, W. X. Bo, and S. Maharjan, “Artificial Intelligence for Vehicle-To-Everything: A Survey,” *IEEE Access*, vol. 7, pp. 10 823–10 843, 2019. DOI: 10.1109/ACCESS.2019.2891073.

[5] J. Wang, Y. Shao, Y. Ge, and R. Yu, “A survey of vehicle to everything (V2X) testing,” *Sensors (Switzerland)*, vol. 19, no. 2, pp. 1–20, 2019. DOI: 10.3390/s19020334.

[6] P. Skobelev, D. Budaev, N. Gusev, and G. Voschuk, “Designing Multi-agent Swarm of UAV for Precise Agriculture BT - Highlights of Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection,” J. Bajo, J. M. Corchado, E. M. Navarro Martínez, *et al.*, Eds., Cham: Springer International Publishing, Jun. 19, 2018, pp. 47–59, ISBN: 978-3-319-94779-2. [Online]. Available: https://www.ebook.de/de/product/33414708/highlights_of_practical_applications_of_agents_multi_agent_systems_and_complexity_the_paams_collection.html.

[7] O. Malaschuk and A. Dyumin, “Intelligent Multi-agent System for Rescue Missions BT - Advanced Technologies in Robotics and Intelligent Systems,” S. Y. Misyurin, V. Arakelian, and A. I. Avetisyan, Eds., Cham: Springer International Publishing, 2020, pp. 89–97, ISBN: 978-3-030-33491-8.

[8] A. M. Mahdy, “Marine wireless sensor networks: Challenges and applications,” *Proceedings - 7th International Conference on Networking, ICN 2008*, pp. 530–535, 2008. DOI: 10.1109/ICN.2008.115.

[9] J. Yick, B. Mukherjee, and D. Ghosal, “Wireless sensor network survey,” *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, Aug. 2008. DOI: 10.1016/j.comnet.2008.04.002.

[10] P. Corke, T. Wark, R. Jurdak, W. Hu, P. Valencia, and D. Moore, “Environmental wireless sensor networks,” *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1903–1917, Nov. 2010. DOI: 10.1109/JPROC.2010.2068530.

[11] C. Albaladejo, P. Sánchez, A. Iborra, F. Soto, J. A. López, and R. Torres, “Wireless sensor networks for oceanographic monitoring: A systematic review,” *Sensors*, vol. 10, no. 7, pp. 6948–6968, Jul. 2010. DOI: 10.3390/s100706948. [Online]. Available: https://www.mdpi.com/journal/sensors.

[12] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: A survey,” *Computer Networks*, vol. 38, no. 4, pp. 393–422, Mar. 2002. DOI: 10.1016/S1389-1286(01)00302-4.

[13] V. Lesser, C. L. Ortiz, and M. Tambe, *Distributed sensor networks : a multiagent perspective*. Kluwer Academic Publishers, 2003, p. 367, ISBN: 9781402074998.

[14] V. C. Gungor and G. P. Hancke, “Industrial wireless sensor networks: Challenges, design principles, and technical approaches,” *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 4258–4265, Oct. 2009. DOI: 10.1109/TIE.2009.2015754.

[15] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, Aug. 2013. DOI: 10.1177/0278364913495721. [Online]. Available: http://journals.sagepub.com/doi/pdf/10.1177/0278364913495721.

[16] J. A. Bagnell and A. Y. Ng, “On Local Rewards and Scaling Distributed Reinforcement Learning,” *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005*, 2005. DOI: 10.5555/2976248.2976260. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2005/file/02180771a9b609a26dcea07f272e141f-Paper.pdf.

[17] J. O. Gutierrez-Garcia and K. M. Sim, “Self-organizing agents for service composition in cloud computing,” *Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010*, pp. 59–66, Nov. 2010. DOI: 10.1109/CloudCom.2010.10.

[18] P. Krivic, P. Skocir, M. Kusek, and G. Jezic, “Agent and Multi-Agent Systems. Technologies and Applications,” *Smart Innovation, Systems and Technologies*, vol. 74, no. January, p22–31, May 2017. DOI: 10.1007/978-3-319-59394-4_3. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-59394-4_3.

[19] J. Parker, “Task allocation for multi-agent systems in dynamic environments,” *12th International Conference on Autonomous Agents and Multiagent Systems 2013, AAMAS 2013*, vol. 2, pp. 1445–1446, 2013.

[20] H. A. A. Al-Rawi, M. A. Ng, and K.-L. A. Yau, “Application of reinforcement learning to routing in distributed wireless networks: a review,” *Artificial Intelligence Review*, vol. 43, no. 3, pp. 381–416, Jan. 2013, ISSN: 0269-2821. DOI: 10.1007/s10462-012-9383-6. [Online]. Available: http://link.springer.com/10.1007/s10462-012-9383-6.

[21] A. Mazrekaj, D. Minarolli, and B. Freisleben, “Distributed resource allocation in cloud computing using multi-agent systems,” *Telfor Journal*, vol. 9, no. 2, pp. 110–115, 2017. DOI: 10.5937/telfor1702110m.

[22] J. Edmondson and D. Schmidt, "Multi-agent distributed adaptive resource allocation (MADARA)," Tech. Rep. 3, 2010, pp. 229–245. DOI: 10.1504/IJCNDS.2010.034946.

[23] C. Zhang, V. Lesser, and P. Shenoy, "A multi-agent learning approach to online distributed resource allocation," in *IJCAI International Joint Conference on Artificial Intelligence*, 2009, ISBN: 9781577354260.

[24] G. Di Marzo Serugendo, N. Foukia, S. Hassas, *et al.*, "Self-Organisation: Paradigms and Applications," in *Engineering Self-Organising Systems*, Springer, Berlin, Heidelberg, 2004, pp. 1–19. DOI: 10.1007/978-3-540-24701-2_1. [Online]. Available: http://link.springer.com/10.1007/978-3-540-24701-2_1.

[25] G. Di Marzo Serugendo, M. Gleizes, and A. Karageorgos, "Self-Organization in Multi-Agent Systems," *The Knowledge Engineering Review*, vol. 20, no. 2, pp. 165–189, Jun. 2005, ISSN: 0269-8889. DOI: 10.1017/S0269888905000494. [Online]. Available: https://doi.org/10.1017/S0269888905000494.

[26] R. Kota, N. Gibbins, and N. R. Jennings, "Self-organising agent organisations," in *The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09) (09/05/09 - 14/05/09)*, Event Dates: 10-15 May, 2009, May 2009, pp. 797–804. [Online]. Available: https://eprints.soton.ac.uk/267071/.

[27] M. Gleizes, "Self-adaptive Complex Systems," in *Multi-Agent Systems*, Springer, Berlin, Heidelberg, 2012, pp. 114–128. DOI: 10.1007/978-3-642-34799-3_8. [Online]. Available: http://link.springer.com/10.1007/978-3-642-34799-3_8.

[28] H. A. Abbas, S. I. Shaheen, and M. H. Amin, "Organization of Multi-Agent Systems : An Overview," *International Journal of Intelligent Information Systems*, vol. 4, no. 3, pp. 46–57, 2015. DOI: 10.11648/j.ijiis.20150403.11. eprint: 1506.09032.

[29] M. Wooldridge and P. Ciancarini, "Agent-Oriented Software Engineering: The State of the Art," in *Agent-Oriented Software Engineering*, Springer Berlin Heidelberg, 2001, pp. 1–28. DOI: 10.1007/3-540-44564-1_1. [Online]. Available: http://link.springer.com/10.1007/3-540-44564-1_1.

[30] R. Calegari, G. Ciatto, V. Mascardi, and A. Omicini, "Logic-based technologies for multi-agent systems: a systematic literature review," *Autonomous Agents and Multi-Agent Systems*, vol. 35, no. 1, p. 1, Oct. 2020, ISSN: 1573-7454. DOI: 10.1007/s10458-020-09478-3. [Online]. Available: https://doi.org/10.1007/s10458-020-09478-3.

[31] R. Brooks, "How to Build Complete Creatures Rather than Isolated Cognitive Simulators, The 22nd carnegie mellon symposium on cognition (carnegie mellon symposia on cognition)," Lawrence Erlbaum, 1987, pp. 225–239, ISBN: 9781315807843.

[32] E. Gat, R. P. Bonnasso, R. Murphy, *et al.*, "On three-layer architectures," *Artificial intelligence and mobile robots*, vol. 195, p. 210, 1998. [Online]. Available: http://flownet.com/ron/papers/tla.pdf.

[33] R. Hartley and F. Pipitone, "Experiments with the subsumption architecture," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, IEEE Comput. Soc. Press, pp. 1652–1658, ISBN: 0-8186-2163-X. DOI: 10.1109/ROBOT.1991.131856. [Online]. Available: http://ieeexplore.ieee.org/document/131856/.

[34] J. D. Velleman and M. E. Bratman, "Intention, Plans, and Practical Reason," *The Philosophical Review*, vol. 100, no. 2, p. 277, Apr. 1991. DOI: 10.2307/2185304. [Online]. Available: http://www.jstor.org/stable/2185304?origin=crossref.

[35] J. Broersen, M. Dastani, J. Hulstijn, Z. Huang, and L. van der Torre, "The BOID Architecture Conflicts Between Beliefs, Obligations, Intentions and Desires," *Proceeding AGENTS '01 Proceedings of the fifth international conference on Autonomous agent*, no. 9-16, May 2001. DOI: 10.1145/375735.375766.

[36] G. Di Marzo Serugendo, M. Gleizes, and A. Karageorgos, "Self-organisation and emergence in mas: An overview.," *Informatica (03505596)*, vol. 30, no. 1, pp. 45–54, 2006.

[37] G. Picard, J. F. Hübner, O. Boissier, and M. Gleizes, "Reorganisation and self-organisation in multi-agent systems," pp. 66–80, 2009.

[38] M. D'Inverno, M. Fisher, A. Lomuscio, *et al.*, "Formalisms for multi-agent systems," *The Knowledge Engineering Review*, vol. 12, no. 3, pp. 315–321, Sep. 1997. DOI: 10.1017/S0269888997003068. [Online]. Available: http://journals.cambridge.org/abstract_S0269888997003068.

[39] M. Luck, M. D'Inverno, and M. Fisher, "Foundations of Multi-Agent Systems: Techniques, Tools and Theory," *The Knowledge Engineering Review*, vol. 13, no. 3, pp. 297–302, 1998. DOI: 10.1017/S0269888998003014. [Online]. Available: http://www.journals.cambridge.org/abstract_S0269888998003014.

[40] P. Valckenaers, H. Van Brussel, M. Kollingbaum, and O. Bochmann, "MULTI-AGENT MANUFACTURING CONTROL USING STIGMERGY," *IFAC Proceedings Volumes*, vol. 35, pp. 67–72, 2002. DOI: 10.3182/20020721-6-ES-1901.00014.

[41] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," pp. 305–319, 2014. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

[42] H. Howard, M. Schwarzkopf, A. Madhavapeddy, and J. Crowcroft, "Raft refloated: Do we have consensus?" *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, 12–21, Jan. 2015, ISSN: 0163-5980. DOI: 10.1145/2723872.2723876. [Online]. Available: https://doi.org/10.1145/2723872.2723876.

[43] A. Lakshman and P. Malik, "Cassandra," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, p. 35, Apr. 2010. DOI: 10.1145/1773912.1773922. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1773912.1773922.

[44] B. Hindman, A. Konwinski, M. Zaharia, *et al.*, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11, Boston, MA: USENIX Association, 2011, 295–308. DOI: 10.5555/1972457.1972488.

[45] Y. Chen and X. Bai, "On robotics applications in service-oriented architecture," *Proceedings - International Conference on Distributed Computing Systems*, pp. 551–556, Jun. 2008, ISSN: 1545-0678. DOI: 10.1109/ICDCS.Workshops.2008.60.

[46] S. Agrawal and R. Kamal, "Computational Orchestrator: A Super Class for Matrix, Robotics and Control System Orchestration," *International Journal of*

Computer Applications, vol. 117, no. 10, pp. 12–19, May 2015. DOI: 10.5120/20588-2923.

[47] A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft, "Safe exploration for reinforcement learning," pp. 143–148, 2008. [Online]. Available: https://www.esann.org/sites/default/files/proceedings/legacy/es2008-36.pdf.

[48] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, pp. 1437–1480, 2015. [Online]. Available: http://www.jmlr.org/papers/volume16/garcia15a/garcia15a.pdf.

[49] M. Pecka, "Safe Exploration Techniques for Reinforcement Learning – An Overview," *Components*, vol. 14, no. 457, pp. 1062–1070, 2014. DOI: 10.1007/978-3-319-13823-7. [Online]. Available: http://discovery.ucl.ac.uk/53087/.

[50] T. Mannucci, E. Van Kampen, C. C. de Visser, and Q. P. Chu, "SHERPA: a safe exploration algorithm for Reinforcement Learning controllers," *AIAA Guidance, Navigation, and Control Conference*, no. February 2017, 2015. DOI: 10.2514/6.2015-1757. [Online]. Available: http://arc.aiaa.org/doi/10.2514/6.2015-1757.

[51] Y. Chevaleyre, P. E. Dunne, U. Endriss, J. Lang, N. Maudet, and J. Rodríguez-Aguilar, "Multiagent resource allocation," *The Knowledge Engineering Review*, vol. 20, no. 2, pp. 143–149, Jun. 2005. DOI: 10.1017/S0269888905000470.

[52] D. Briola and V. Mascardi, "Multi agent resource allocation: A comparison of five negotiation protocols," *CEUR Workshop Proceedings*, vol. 741, pp. 95–104, 2011. [Online]. Available: https://hdl.handle.net/11383/2118495.

[53] Y. Chevaleyre, P. E. Dunne, U. Endriss, *et al.*, "Issues in Multiagent Resource Allocation," vol. 30, no. 1, pp. 3–31, 2006, ISSN: 0350-5596.

[54] K. Fischer, M. Schillo, and J. Siekmann, "Holonic Multiagent Systems: A Foundation for the Organisation of Multiagent Systems," in *Holonic and Multi-Agent Systems for Manufacturing*, Springer, Berlin, Heidelberg, 2003, pp. 71–80. DOI: 10.1007/978-3-540-45185-3_7. [Online]. Available: http://link.springer.com/10.1007/978-3-540-45185-3{_}7.

[55] S. Rodriguez, V. Hilaire, N. Gaud, S. Galland, and A. Koukam, "Holonic Multi-Agent Systems," 2011, pp. 251–279. DOI: 10.1007/978-3-642-17348-6_11. [Online]. Available: http://link.springer.com/10.1007/978-3-642-17348-6{_}11.

[56] M. Wang, Q. Li, and Y. Lin, "An Organizational Structure and Self-adaptive Mechanism for Holonic Multi-Agent Systems," *IEEE Access*, p. 1, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3014694. [Online]. Available: https://ieeexplore.ieee.org/document/9160943/.

[57] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, no. 2, pp. 128–135, 2010. DOI: 10.1049/iet-its.2009.0070.

[58] J. Cui, Y. Liu, and A. Nallanathan, "Multi-Agent Reinforcement Learning-Based Resource Allocation for UAV Networks," *IEEE Transactions on Wireless Communications*, vol. 19, no. 2, pp. 729–743, Feb. 2020. DOI: 10.1109/TWC.2019.2935201. arXiv: 1810.10408.

[59] S. Z. Jafarzadeh and M. H. Y. Moghaddam, "Design of energy-aware QoS routing protocol in wireless sensor networks using reinforcement learning," *Cana-*

dian Conference on Electrical and Computer Engineering, pp. 1–5, May 2014. DOI: 10.1109/CCECE.2014.6900988.

[60] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: Towards a Fully Automated Workflow," c, 2011, p. 8, ISBN: 9781612081342. [Online]. Available: https://www.cl.cam.ac.uk/~ey204/teaching/ACS/R212_2015_2016/papers/dutreilh_icas_2011.pdf.

[61] F. R. Yu, V. W. S. Wong, and V. C. M. Leung, "A new QoS provisioning method for adaptive multimedia in wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 3, pp. 1899–1909, 2008. DOI: 10.1109/TVT.2007.907023.

[62] T. Nothdurft, P. Hecker, T. Frankiewicz, J. Gačnik, and F. Köster, "Reliable information aggregation and exchange for autonomous vehicles," *IEEE Vehicular Technology Conference*, 2011. DOI: 10.1109/VETECF.2011.6092937.

[63] Y. Xie, H. Zhang, N. H. Gartner, and T. Arsava, "Collaborative merging strategy for freeway ramp operations in a connected and autonomous vehicles environment," *Journal of Intelligent Transportation Systems: Technology, Planning, and Operations*, vol. 21, no. 2, pp. 136–147, 2017. DOI: 10.1080/15472450.2016.1248288. [Online]. Available: https://doi.org/10.1080/15472450.2016.1248288.

[64] B. Schünemann, J. W. Wedel, and I. Radusch, "V2X-based traffic congestion recognition and avoidance," *Tamkang Journal of Science and Engineering*, vol. 13, no. 1, pp. 63–70, 2010. DOI: 10.6180/jase.2010.13.1.07.

[65] M. De Weerdt and B. Clement, "Introduction to Planning in Multiagent Systems," *Multiagent Grid Syst.*, vol. 5, no. 4, M. de Weerdt and B. Clement, Eds., pp. 345–355, Dec. 2009. DOI: 10.3233/mgs-2009-0133. [Online]. Available: http://ww.nextgenerationinfrastructures.eu/catalog/file/534037/MAGS5_09_DeWeerdt.pdf.

[66] J. Van Der Horst and J. Noble, "Distributed and centralized task allocation: When and where to use them," *Proceedings - 2010 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop, SASOW 2010*, pp. 1–8, 2010. DOI: 10.1109/SASOW.2010.8.

[67] X. Jia and M. Q. H. Meng, "A survey and analysis of task allocation algorithms in multi-robot systems," *2013 IEEE International Conference on Robotics and Biomimetics, ROBIO 2013*, no. December, pp. 2280–2285, Dec. 2013. DOI: 10.1109/ROBIO.2013.6739809.

[68] Y. Jiang, "A Survey of Task Allocation and Load Balancing in Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 585–599, Feb. 2016. DOI: 10.1109/TPDS.2015.2407900.

[69] C. Zhang, V. Lesser, and P. Shenoy, "A multi-agent learning approach to online distributed resource allocation," 2009.

[70] N. Krothapalli and A. V. Deshmukh, "Distributed Task Allocation in Multi – Agent Systems," *Proceedings of the Institute of Industrial Engineers*, 2002.

[71] L. Buşoniu, R. Babuška, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, Mar. 2008. DOI: 10.1109/TSMCC.2007.913919.

[72] V. Singhal and D. Dahiya, "Distributed task allocation in dynamic multi-agent system," *International Conference on Computing, Communication and Automation, ICCCA 2015*, pp. 643–648, 2015. DOI: 10.1109/CCAA.2015.7148452.

[73] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent Reinforcement Learning: An Overview," *Proceedings of the 2nd International Conference on Multi-Agent Systems*, vol. 19, pp. 183–221, 2010. DOI: 10.1007/978-3-642-14435-6_7. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-14435-6_7.

[74] K. Tuyls and G. Weiss, "Multiagent Learning:Basics, Challanges, and Prospects," *AI Magazine*, pp. 41–52, 2012, ISSN: 0738-4602.

[75] D. Abel, D. E. Hershkowitz, and M. L. Littman, "Near optimal behavior via approximate state abstraction," *33rd International Conference on Machine Learning, ICML 2016*, vol. 6, pp. 4287–4295, Jan. 15, 2017. DOI: 10.48550/ARXIV.1701.04113. arXiv: 1701.04113 [cs.LG].

[76] D. C. L. Vieira, P. J. L. Adeodato, and P. M. Gonçalves, "A temporal difference GNG-based approach for the state space quantization in reinforcement learning environments," *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, pp. 561–568, 2013. DOI: 10.1109/ICTAI.2013.89.

[77] A. Notsu, K. Yasuda, S. Ubukata, and K. Honda, "Online state space generation by a growing self-organizing map and differential learning for reinforcement learning," *Applied Soft Computing*, vol. 97, p. 106 723, 2020, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2020.106723. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S156849462030661X.

[78] M. Gueriau, N. Cardozo, and I. Dusparic, "Constructivist Approach to State Space Adaptation in Reinforcement Learning," *International Conference on Self-Adaptive and Self-Organizing Systems, SASO*, vol. 2019-June, pp. 52–61, Jun. 2019. DOI: 10.1109/SASO.2019.00016.

[79] I. Dusparic and N. Cardozo, *Adaptation to Unknown Situations as the Holy Grail of Learning-Based Self-Adaptive Systems: Research Directions*, Mar. 11, 2021. DOI: 10.48550/ARXIV.2103.06908. arXiv: 2103.06908 [cs.AI].

[80] R. McFarlane, "A Survey of Exploration Methods in Reinforcement Learning," *McGill University*, pp. 1–10, 2003. [Online]. Available: https://www.cs.mcgill.ca/~cs526/roger.pdf.

[81] S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, and D. Precup, *A Survey of Exploration Methods in Reinforcement Learning*, Sep. 1, 2021. DOI: 10.48550/ARXIV.2109.00157. arXiv: 2109.00157 [cs.LG]. [Online]. Available: http://arxiv.org/abs/2109.00157.

[82] S. B. Thrun, "Efficient Exploration In Reinforcement Learning," Carnegie Mellon University, USA, Tech. Rep., 1992. DOI: 10.5555/865072.

[83] M. Wunder, M. L. Littman, and M. Babes-Vroman, "Classes of Multiagent Q-learning Dynamics with epsilon-greedy Exploration," in *ICML*, 2010, pp. 1167–1174.

[84] F. Stulp, "Adaptive exploration for continual reinforcement learning," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 1631–1636. DOI: 10.1109/IROS.2012.6385818.

[85] Z. Xu, X. Chen, L. Cao, and C. Li, "A study of count-based exploration and bonus for reinforcement learning," in *2017 IEEE 2nd International Conference*

on Cloud Computing and Big Data Analysis (ICCCBDA), 2017, pp. 425–429. DOI: 10.1109/ICCCBDA.2017.7951951.

[86] P. Dayan, "Improving Generalization for Temporal Difference Learning: The Successor Representation," *Neural Computation*, vol. 5, no. 4, pp. 613–624, Jul. 1993, ISSN: 0899-7667. DOI: 10.1162/neco.1993.5.4.613.

[87] M. C. Machado, A. Barreto, and D. Precup, *Temporal Abstraction in Reinforcement Learning with the Successor Representation*, 2021. arXiv: 2110.05740 [cs.LG].

[88] A. Pacchiano, P. Ball, J. Parker-holder, K. Choromanski, and S. Roberts, *On Optimism in Model-Based Reinforcement*. arXiv: arXiv:2006.11911v1.

[89] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *J. Artif. Int. Res.*, vol. 4, no. 1, pp. 237–285, May 1996, ISSN: 1076-9757.

[90] J. Tarbouriech, E. Garcelon, M. Valko, M. Pirotta, and A. Lazaric, *No-Regret Exploration in Goal-Oriented Reinforcement Learning*, 2020. arXiv: 1912.03517 [stat.ML].

[91] A. Modi and A. Tewari, "No-regret Exploration in Contextual Reinforcement Learning," in *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, J. Peters and D. Sontag, Eds., ser. Proceedings of Machine Learning Research, vol. 124, PMLR, 2020, pp. 829–838. eprint: 1903.06187. [Online]. Available: https://proceedings.mlr.press/v124/modi20a.html.

[92] P. Oudeyer and F. Kaplan, "How can we define intrinsic motivation?" *8th International Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems, Lund University Cognitive Studies*, no. July, pp. 1–10, 2013.

[93] T. Blau, L. Ott, and F. Ramos, *Bayesian Curiosity for Efficient Exploration in Reinforcement Learning*, Nov. 20, 2019. DOI: 10.48550/ARXIV.1911.08701. arXiv: 1911.08701 [cs.LG].

[94] N. Bougie and R. Ichise, *Intrinsically Motivated Lifelong Exploration in Reinforcement Learning*. Springer International Publishing, 2021, pp. 109–120. DOI: 10.1007/978-3-030-73113-7_10.

[95] R. Patrascu and D. Stacey, "Adaptive exploration in reinforcement learning," in *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, vol. 4, IEEE, 1999, pp. 2276–2281. DOI: 10.1109/IJCNN.1999.833417.

[96] G. Sokar, D. C. Mocanu, and M. Pechenizkiy, *Addressing the Stability-Plasticity Dilemma via Knowledge-Aware Continual Learning*, 2021. arXiv: 2110.05329 [cs.LG].

[97] C. V. Nguyen, A. Achille, M. Lam, T. Hassner, V. Mahadevan, and S. Soatto, *Toward Understanding Catastrophic Forgetting in Continual Learning*, 2019. arXiv: 1908.01091. [Online]. Available: http://arxiv.org/abs/1908.01091.

[98] M. Pieters and M. A. Wiering, "Q-learning with experience replay in a dynamic environment," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016, pp. 1–8. DOI: 10.1109/SSCI.2016.7849368.

[99] D. Isele and A. Cosgun, "Selective Experience Replay for Lifelong Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Feb. 28, 2018. DOI: 10.48550/ARXIV.1802.10269. arXiv: 1802.10269 [cs.AI]. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11595.

[100] D. Rolnick, A. Ahuja, J. Schwarz, T. P. Lillicrap, and G. Wayne, *Experience Replay for Continual Learning*, 2019. arXiv: 1811.11682 [cs.LG].

[101] T. Zhang, X. Wang, B. Liang, and B. Yuan, *Catastrophic Interference in Reinforcement Learning: A Solution Based on Context Division and Knowledge Distillation*, 2021. arXiv: 2109.00525. [Online]. Available: http://arxiv.org/abs/2109.00525.

[102] Y. Lo and S. Ghiassian, *Overcoming Catastrophic Interference in Online Reinforcement Learning with Dynamic Self-Organizing Maps*, Oct. 29, 2019. DOI: 10.48550/ARXIV.1910.13213. arXiv: 1910.13213 [cs.AI].

[103] F. S. Melo and M. Veloso, "Learning of Coordination: Exploiting Sparse Interactions in Multiagent Systems," *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, pp. 773–780, 2009. DOI: 10.1145/1558109.1558118.

[104] Y. M. De Hauwere, P. Vrancx, and A. Nowé, "Learning multi-agent state space representations," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, ser. AAMAS '10, Toronto, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2010, 715–722, ISBN: 9780982657119.

[105] Y. M. De Hauwere, P. Vrancx, and A. Nowé, "Solving sparse delayed coordination problems in multi-agent reinforcement learning, Aamas 2011 international workshop, ala 2011, taipei, taiwan, may 2, 2011, revised selected papers," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7113 LNAI, Springer-Verlag, 2012, pp. 114–133, ISBN: 9783642284984. DOI: 10.1007/978-3-642-28499-1_8. [Online]. Available: http://link.springer.com/10.1007/978-3-642-28499-1_8.

[106] C. Sun, W. Liu, and L. Dong, "Reinforcement Learning With Task Decomposition for Cooperative Multiagent Systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 5, pp. 2054–2065, 2021. DOI: 10.1109/TNNLS.2020.2996209.

[107] C. Yang and S. Chen, "Survey on Modeling and Controlling of Welding Robot Systems Based on Multi-agent BT - Robotic Welding, Intelligence and Automation," T.-J. Tarn, S.-B. Chen, and G. Fang, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 107–113, ISBN: 978-3-642-19959-2.

[108] S. Hwangbo, J. Jeon, and S. Park, "Self-Powered Wireless Ocean Monitoring Systems," c, 2012, pp. 334–337, ISBN: 9781612082073.

[109] C. Shyalika, T. Silva, and A. Karunananda, "Reinforcement Learning in Dynamic Task Scheduling: A Review," *SN Computer Science*, vol. 1, no. 6, pp. 1–17, 2020, ISSN: 2662-995X. DOI: 10.1007/s42979-020-00326-5. [Online]. Available: https://doi.org/10.1007/s42979-020-00326-5.

[110] G. Rizzo, M. R. Palattella, T. Braun, and T. Engel, "Content and context aware strategies for QoS support in VANETs," *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, vol. 2016-May, pp. 717–723, 2016. DOI: 10.1109/AINA.2016.85.

[111] M. H. Eiza, T. Owens, Q. Ni, and Q. Shi, "Situation-aware QoS routing algorithm for vehicular ad hoc networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 12, pp. 5520–5535, Dec. 2015. DOI: 10.1109/TVT.2015.2485305.

[112] C. Belagal Math, H. Li, S. Heemstra De Groot, and I. G. Niemegeers, “V2X Application-Reliability Analysis of Data-Rate and Message-Rate Congestion Control Algorithms,” *IEEE Communications Letters*, vol. 21, no. 6, pp. 1285–1288, Jun. 2017. DOI: 10.1109/LCOMM.2017.2675899.

[113] J. Feng, Z. Liu, C. Wu, and Y. Ji, “AVE: Autonomous vehicular edge computing framework with ACO-based scheduling,” *IEEE Transactions on Vehicular Technology*, vol. 66, no. 12, pp. 10 660–10 675, Dec. 2017. DOI: 10.1109/TVT.2017.2714704.

[114] S. Xu, C. Guo, R. Q. Hu, and Y. Qian, “Multi-Agent Deep Reinforcement Learning enabled Computation Resource Allocation in a Vehicular Cloud Network,” pp. 1–14, 2020. arXiv: 2008.06464. [Online]. Available: http://arxiv.org/abs/2008.06464.

[115] C. Beam and A. Segev, “Automated negotiations: A survey of the state of the art,” *Wirtschaftsinformatik*, vol. 39, no. 3, pp. 263–268, 1997.

[116] S. Fatima and M. Wooldridge, “Adaptive Task and Resource Allocation in Multi-Agent Systems,” *Proceedings of the Fifth International Conference on Autonomous Agents : Montreal, Canada, May 28-June 1, 2001*, p. 666, May 2001. DOI: https://doi.org/10.1145/375735.376439.

[117] R. G. Smith, “The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver,” *IEEE TRANSACTIONS ON COMPUTERS*, vol. 29, no. 12, 1980.

[118] E. Bozdag, “A survey of extensions to the contract net protocol,” *Technical report, CiteSeerX-Scientific Literature Digital Library and Search Engine*, no. 1119869, 2008.

[119] B. Far, T. Wanyama, and S. O. Soueina, “A Negotiation Model for Large Scale Multi-Agent Systems,” in *2006 IEEE International Conference on Information Reuse and Integration*, IEEE, Sep. 2006, pp. 589–594. DOI: 10.1109/IRI.2006.252480.

[120] S. Vries de and R. V. Vohra, “Combinatorial Auctions : A Survey,” *Journal on Computing*, vol. 15, no. 3, pp. 284–309, 1998.

[121] D. C. Parkes and L. H. Ungar, “Iterative combinatorial auctions: Theory and practice,” *Proceedings of the 17th National Conference on Artificial Intelligence AAAI-00*, pp. 74–81, 2000. [Online]. Available: https://eprints.kfupm.edu.sa/47192/1/47192.pdf.

[122] C. Claus and C. Boutilier, “The dynamics of reinforcement learning in cooperative multiagent systems,” in *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, ser. AAAI ’98/IAAI ’98, Madison, Wisconsin, USA: American Association for Artificial Intelligence, 1998, 746–752, ISBN: 0262510987. DOI: 10.5555/295240.295800.

[123] M. McGlohon and S. Sen, “Learning to cooperate in multi-agent systems by combining Q-learning and evolutionary strategy,” *International Journal on Lateral Computing*, vol. 1, no. 2, pp. 58–64, 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.6511&rep=rep1&type=pdf.

[124] J. E. Posor, L. Belzner, and A. Knapp, “Joint Action Learning for Multi-Agent Cooperation using Recurrent Reinforcement Learning,” *Digitale Welt*, vol. 4, no. 1, pp. 79–84, Dec. 2019. DOI: 10.1007/s42354-019-0239-y.

[125] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson, *Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning*, 2018. arXiv: 1803.11485 [cs.LG].

[126] M. Zhou, Y. Chen, Y. Wen, *et al.*, "Factorized Q-learning for large-scale multi-agent systems," *ACM International Conference Proceeding Series*, 2019. DOI: 10.1145/3356464.3357707. arXiv: arXiv:1809.03738v4.

[127] L. Pan, T. Rashid, B. Peng, L. Huang, and S. Whiteson, "Regularized softmax deep multi-agent q-learning," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 1365–1377. eprint: 2103.11883. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/0a113ef6b61820daa5611c870ed8d5ee-Paper.pdf.

[128] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," 2000.

[129] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Hysteretic Q-learning : an algorithm for Decentralized Reinforcement Learning in Cooperative Multi-Agent Teams," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Oct. 2007, pp. 64–69. DOI: 10.1109/IROS.2007.4399095.

[130] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multi-agent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, Oct. 2019. DOI: 10.1007/s10458-019-09421-1.

[131] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[132] C. S. de Witt, T. Gupta, D. Makoviichuk, *et al.*, *Is independent learning all you need in the starcraft multi-agent challenge?* Nov. 18, 2020. DOI: 10.48550/ARXIV.2011.09533. arXiv: 2011.09533 [cs.AI].

[133] C. Yu, A. Velu, E. Vinitsky, *et al.*, *The surprising effectiveness of ppo in cooperative, multi-agent games*, 2022. arXiv: 2103.01955 [cs.LG].

[134] P. K. Sharma, E. G. Zaroukian, R. Fernandez, A. Basak, and D. E. Asher, "Survey of recent multi-agent reinforcement learning algorithms utilizing centralized training," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, T. Pham, L. Solomon, and M. E. Hohil, Eds., SPIE, Apr. 2021. DOI: 10.1117/12.2585808. eprint: 2107.14316. [Online]. Available: https://doi.org/10.1117%2F12.2585808.

[135] X. Lyu, Y. Xiao, B. Daley, and C. Amato, *Contrasting centralized and decentralized critics in multi-agent reinforcement learning*, 2021. arXiv: 2102.04402 [cs.LG].

[136] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," Jun. 2017. arXiv: 1706.02275. [Online]. Available: http://arxiv.org/abs/1706.02275.

[137] M. E. Harmon and S. S. Harmon, "Reinforcement learning: a tutorial," *INFORMS Journal on Computing*, vol. 45433, no. 2, pp. 237–285, May 2009. DOI: 10.21236/ada323194. [Online]. Available: https://robotics.ee.uwa.edu.au/courses/robotics/reading/Reinforcement-Harmon2000.pdf.

[138] P. Dayan and C. Watkins, "Reinforcement Learning," Jul. 2002. DOI: https://doi.org/10.1002/0471214426.pas0303. [Online]. Available: http://www.gatsby.ucl.ac.uk/~dayan/papers/dw01.pdf.

[139] R. S. Sutton and A. G. Barto, *Reinforcement learning : an introduction, an introduction*. MIT Press, 1998, p. 322, ISBN: 9780262193986.

[140] M. P. Deisenroth, G. Neumann, and J. Peters, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, vol. 22, no. 1, pp. 1–141, 2011. DOI: 10.1561/2300000021. [Online]. Available: https://spiral.imperial.ac.uk:8443/bitstream/10044/1/12051/7/fnt_corrected_2014-8-22.pdf.

[141] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," *Proceedings of International Conference on Robotics and Automation*, vol. 4, no. April, pp. 3557–3564, 1997. DOI: 10.1109/ROBOT.1997.606886.

[142] Q. J. M. Huys, A. Cruickshank, and P. Seriès, "Reward-based learning, model-based and model-free," pp. 1–10, 2014. DOI: https://doi.org/10.1007/978-1-4614-7320-6_674-1. [Online]. Available: https://www.quentinhuys.com/pub/HuysEa14-ModelBasedModelFree.pdf.

[143] P. Dayan and Y. Niv, "Reinforcement learning: The Good, The Bad and The Ugly," *Current Opinion in Neurobiology*, vol. 18, no. 2, pp. 1–12, Apr. 2008. DOI: 10.1016/j.conb.2008.08.003. [Online]. Available: https://www.princeton.edu/~yael/Publications/DayanNiv2008.pdf.

[144] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," *CoRR*, vol. abs/1702.08892, 2017. arXiv: 1702.08892. [Online]. Available: http://arxiv.org/abs/1702.08892.

[145] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12, MIT Press, 1999. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[146] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM Journal on Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, 2003. DOI: 10.1137/S0363012901385691.

[147] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, 1291–1307, Nov. 2012, ISSN: 1094-6977. DOI: 10.1109/TSMCC.2012.2218595. [Online]. Available: https://doi.org/10.1109/TSMCC.2012.2218595.

[148] T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-free reinforcement learning with continuous action in practice," in *2012 American Control Conference (ACC)*, IEEE, Jun. 2012, pp. 2177–2182. DOI: 10.1109/ACC.2012.6315022.

[149] Y. Xiao, X. Lyu, and C. Amato, "Local Advantage Actor-Critic for Robust Multi-Agent Deep Reinforcement Learning," *2021 International Symposium on Multi-Robot and Multi-Agent Systems, MRS 2021*, pp. 155–163, 2021. DOI: 10.1109/MRS50823.2021.9620607. arXiv: 2110.08642.

[150] S. G. Barnes, "Decentralized Multi-Agent Advantage Actor-Critic Decentralized Multi-Agent Advantage Actor-Critic," pp. 1–6, Feb. 2022. DOI: 10.36227/techrxiv.19166384.v1.

[151] P. Trivedi and N. Hemachandra, "Multi-Agent Natural Actor-Critic Reinforcement Learning Algorithms," *Dynamic Games and Applications*, vol. 13, no. 1,

pp. 25–55, Jun. 2022, ISSN: 2153-0793. DOI: 10.1007/s13235-022-00449-9. [Online]. Available: https://doi.org/10.1007/s13235-022-00449-9.

[152] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A Brief Survey of Deep Reinforcement Learning," *IEEE SIGNAL PROCESSING MAGAZINE*, vol. 34, no. Special Issue on Deep Learning for Image Understanding (ArXiv extended versions), pp. 26–38, Aug. 19, 2017. DOI: 10.1109/msp.2017.2743240. arXiv: 1708.05866 [cs.LG]. [Online]. Available: https://arxiv.org/pdf/1708.05866.pdf.

[153] X. Wang, S. Wang, X. Liang, *et al.*, "Deep reinforcement learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2022. DOI: 10.1109/TNNLS.2022.3207346.

[154] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. DOI: 10.1038/nature14236. [Online]. Available: http://dx.doi.org/10.1038/nature14236.

[155] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, *Trust region policy optimization*, 2017. arXiv: 1502.05477 [cs.LG].

[156] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, *Continuous control with deep reinforcement learning*, Sep. 9, 2015. DOI: 10.48550/ARXIV.1509.02971. arXiv: 1509.02971 [cs.LG].

[157] Baeldung. "Epsilon-Greedy Q-learning." (2021), [Online]. Available: https://www.baeldung.com/cs/epsilon-greedy-q-learning.

[158] M. Tokic, in *KI 2010: Advances in Artificial Intelligence*, R. Dillmann, J. Beyerer, U. D. Hanebeck, and T. Schultz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 203–210, ISBN: 978-3-642-16111-7.

[159] N. Cesa-Bianchi, C. Gentile, G. Lugosi, and G. Neu, *Boltzmann Exploration Done Right*, May 29, 2017. DOI: 10.48550/ARXIV.1705.10257. arXiv: 1705.10257 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1705.10257.

[160] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, *Curiosity-driven Exploration by Self-supervised Prediction*, 2017. DOI: 10.48550/ARXIV.1705.05363. eprint: 1705.05363. [Online]. Available: https://arxiv.org/abs/1705.05363.

[161] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, *Large-Scale Study of Curiosity-Driven Learning*, 2018. DOI: 10.48550/ARXIV.1808.04355. eprint: 1808.04355. [Online]. Available: https://arxiv.org/abs/1808.04355.

[162] S. Iqbal and F. Sha, "Coordinated Exploration via Intrinsic Rewards for Multi-Agent Reinforcement Learning," May 28, 2019. DOI: 10.48550/ARXIV.1905.12127. arXiv: 1905.12127 [cs.LG]. [Online]. Available: http://arxiv.org/abs/1905.12127.

[163] S. W. Carden, J. O. Lindborg, and Z. Utic, "Improved Exploration in Reinforcement Learning Environments with Low-Discrepancy Action Selection," *AppliedMath*, vol. 2, no. 2, pp. 234–246, May 2022, ISSN: 2673-9909. DOI: 10.3390/appliedmath2020014. [Online]. Available: https://www.mdpi.com/2673-9909/2/2/14.

[164] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, vol. 3, pp. 9–44, 1988. [Online]. Available: https://webdocs.cs.ualberta.ca~sutton/papers/sutton-88-with-erratum.pdf.

[165] P. Y. Glorennec, “Reinforcement learning: An overview,” in *Proceedings European Symposium on Intelligent Techniques (ESIT-00), Aachen, Germany*, Citeseer, 2000, pp. 14–15. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b373b0c6e3b4fef4ac0534965708fc382343f8dc.

[166] S. P. Singh and R. S. Sutton, “Reinforcement learning with replacing eligibility traces,” *Machine Learning*, vol. 22, no. 1-3, pp. 123–158, 1996. DOI: 10.1007/BF00114726. [Online]. Available: https://pdfs.semanticscholar.org/e8b7/d26dba68cca0113b7ba581bda61bccc6df66.pdf.

[167] B. Tanner and R. S. Sutton, “Td lambda networks,” *ICML 2005 - Proceedings of the 22nd International Conference on Machine Learning*, pp. 889–896, 2005. DOI: {https://doi.org/10.1145/1102351.1102463}. [Online]. Available: https://icml.cc/Conferences/2005/proceedings/papers/112_TDLambdaNetworks_TannerSutton.pdf.

[168] C. J. C. H. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

[169] R. Babuška, L. Buoniu, B. De Schutter, R. Babuška, L. Buoniu, and B. De Schutter, “Reinforcement learning for multi-agent systems,” no. 06-041, 2006, Paper for a keynote presentation at the *11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2006),* Prague, Czech Republic, Sept. 2006. [Online]. Available: http://pub.deschutter.info/abs/06_041.html.

[170] H. Zhang and T. Yu, *Taxonomy of Reinforcement Learning Algorithms*, H. Dong, Z. Ding, and S. Zhang, Eds. Singapore: Springer Singapore, 2020, pp. 125–133, ISBN: 978-981-15-4095-0. DOI: 10.1007/978-981-15-4095-0_3. [Online]. Available: https://doi.org/10.1007/978-981-15-4095-0_3.

[171] Wikipedia contributors, *Reinforcement learning, Comparison of reinforcement learning algorithms*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Reinforcement_learning.

[172] J. Achiam, “Spinning Up in Deep Reinforcement Learning,” 2018. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[173] M. L. Littman, “Value-function reinforcement learning in Markov games,” *Journal of Cognitive Systems Research*, vol. 2, pp. 55–66, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1389041701000158.

[174] X. Wang and T. Sandholm, “Reinforcement Learning to Play an Optimal Nash Equilibrium in Team Markov Games,” 2002. [Online]. Available: https://papers.nips.cc/paper/2171-reinforcement-learning-to-play-an-optimal-nash-equilibrium-in-team-markov-games.pdf.

[175] A. Greenwald and K. Hall, “Correlated-q learning, August 21-24, 2003 washington, dc usa,” in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ser. ICML’03, Washington, DC, USA: AAAI Press, 2003, 242–249, ISBN: 1577351894.

[176] H. van Hasselt, A. Guez, and D. Silver, *Deep reinforcement learning with double q-learning*, 2015. arXiv: 1509.06461 [cs.LG].

[177] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, *Dueling network architectures for deep reinforcement learning*, 2016. arXiv: 1511.06581 [cs.LG].

[178] M. Bowling and M. Veloso, "Multiagent learning using a variable learning rate," *Artificial Intelligence*, vol. 136, no. 2, pp. 215–250, Apr. 2002, ISSN: 0004-3702. DOI: 10.1016/S0004-3702(02)00121-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370202001212.

[179] N. Suematsu and A. Hayashi, "A multiagent reinforcement learning algorithm using extended optimal response," in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems part 1 - AAMAS '02*, New York, New York, USA: ACM Press, 2002, p. 370, ISBN: 1581134800. DOI: 10.1145/544741.544831. [Online]. Available: http://portal.acm.org/citation.cfm?doid=544741.544831.

[180] M. Bowling, "Convergence and no-regret in multiagent learning," in *Proceedings of the 17th International Conference on Neural Information Processing Systems*, ser. NIPS'04, Cambridge, MA, USA: MIT Press, 2004, 209–216. DOI: 10.5555/2976040.2976067.

[181] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, arXiv, May 24, 2017. DOI: 10.48550/ARXIV.1705.08926. arXiv: 1705.08926 [cs.AI].

[182] K. Su and Z. Lu, *Decentralized policy optimization*, 2022. arXiv: 2211.03032 [cs.LG].

[183] M. Bowling and M. Veloso, "Rational and convergent learning in stochastic games, 17th international joint conference on artificial intelligence 2-volume & cd set (international joint conference on artificial intelligence//proceedings)," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'01, Lawrence Erlbaum Associates Ltd, vol. 17, Seattle, WA, USA: Morgan Kaufmann Publishers Inc., 2001, 1021–1026, ISBN: 1558608125.

[184] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ser. ICML'03, Washington, DC, USA: AAAI Press, 2003, 928–935, ISBN: 1577351894.

[185] F. S. Melo, S. P. Meyn, and M. I. Ribeiro, "An analysis of reinforcement learning with function approximation," *Proceedings of the 25th International Conference on Machine Learning*, pp. 664–671, 2008. [Online]. Available: http://icml2008.cs.helsinki.fi/papers/652.pdf.

[186] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *J. Mach. Learn. Res.*, vol. 21, no. 1, Jan. 2020, ISSN: 1532-4435. eprint: 2003.04960.

[187] D. Zhang, W. Bao, W. Liang, G. Wu, and J. Cao, "A curriculum learning based multi-agent reinforcement learning method for realtime strategy game," in *2022 8th International Conference on Big Data and Information Analytics (BigDIA)*, 2022, pp. 447–452. DOI: 10.1109/BigDIA56350.2022.9874056.

[188] A. Rodríguez, R. Parr, and D. Koller, "Reinforcement learning using approximate belief states," *Advances in Neural Information Processing Systems*, pp. 1036–1042, 2000.

[189] W. Curran, T. Brys, D. Aha, M. Taylor, and W. D. Smart, "Dimensionality Reduced Reinforcement Learning for Assistive Robots," *Papers from the 2016 AAAI Fall Symposium*, 2016. [Online]. Available: http://www.aaai.org/ocs/index.php/FSS/FSS16/paper/viewFile/14076/13660.

[190] W. Li, M. Gauci, and R. Gross, “Turing learning: a metric-free approach to inferring behavior and its application to swarms,” pp. 1–37, 2016. arXiv: 1603.04904. [Online]. Available: http://arxiv.org/abs/1603.04904.

[191] M. Kearns, Y. Mansour, and A. Ng, “A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes,” *Machine Learning*, vol. 49, no. 2/3, pp. 193–208, 2002. DOI: https://doi.org/10.1023/A:1017932429737.

[192] C. F. Alves, E. L. Colombini, and C. H. C. Ribeiro, “Sparse sampling action values initialized by a compact representation technique,” *Proceedings of The 7th International Conference on Intelligent Systems Design and Applications, ISDA 2007*, pp. 729–734, Oct. 2007. DOI: 10.1109/ISDA.2007.4389694.

[193] T. M. Moldovan and P. Abbeel, “Safe Exploration in Markov Decision Processes,” May 22, 2012. DOI: 10.48550/ARXIV.1205.4810. arXiv: 1205.4810 [cs.LG]. [Online]. Available: https://arxiv.org/pdf/1205.4810.pdf.

[194] S. Barrett, M. E. Taylor, and P. Stone, “Transfer Learning for Reinforcement Learning on a Physical Robot,” in *Adaptive Agents and Multi-Agent Systems*, 2010. [Online]. Available: https://www.cs.utexas.edu/{~}pstone/Papers/bib2html-links/AAMASWS10-barrett.pdf.

[195] L. Orseau and S. Armstrong, “Safely interruptible agents,” *32nd Conference on Uncertainty in Artificial Intelligence 2016, UAI 2016*, pp. 557–566, 2016.

[196] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, “Concrete Problems in AI Safety,” *arXiv*, pp. 1–29, Jun. 21, 2016. DOI: 10.48550/ARXIV.1606.06565. arXiv: 1606.06565 [cs.AI]. [Online]. Available: http://arxiv.org/abs/1606.06565.

[197] S. Armstrong, J. Leike, L. Orseau, and S. Legg, “Pitfalls of learning a reward function online,” *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2021-January, pp. 1592–1600, Jul. 2020. DOI: 10.24963/ijcai.2020/221. arXiv: 2004.13654.

[198] L. Zhou, P. Yang, C. Chen, and Y. Gao, “Multi-agent Reinforcement Learning with Sparse Interactions by Negotiation and Knowledge Transfer,” 2016. arXiv: arXiv:1508.05328v2. [Online]. Available: https://arxiv.org/pdf/1508.05328.pdf.

[199] J. Li and P. Gajane, *Curiosity-driven exploration in sparse-reward multi-agent reinforcement learning*, Feb. 21, 2023. DOI: 10.48550/ARXIV.2302.10825. arXiv: 2302.10825 [cs.AI].

[200] J. Chen, Y. Zhang, Y. Xu, *et al.*, *Variational automatic curriculum learning for sparse-reward cooperative multi-agent problems*, Nov. 8, 2021. DOI: 10.48550/ARXIV.2111.04613. arXiv: 2111.04613 [cs.LG].

[201] C. Packer, P. Abbeel, and J. E. Gonzalez, *Hindsight task relabelling: Experience replay for sparse reward meta-rl*, 2021. arXiv: 2112.00901 [cs.AI].

[202] S. Zhang and R. S. Sutton, *A deeper look at experience replay*, 2017. DOI: https://doi.org/10.48550/arXiv.1712.01275. arXiv: 1712.01275 [cs.LG].

[203] W. Fedus, P. Ramachandran, R. Agarwal, *et al.*, “Revisiting fundamentals of experience replay,” in *Proceedings of the 37th International Conference on Machine Learning*, I. I. I. Hal Daumé and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, Jul. 13, 2020, pp. 3061–3071. DOI: 10.48550/ARXIV.2007.06700. arXiv: 2007.06700 [cs.LG]. [Online]. Available: https://proceedings.mlr.press/v119/fedus20a.html.

[204] J. Foerster, N. Nardelli, G. Farquhar, *et al.*, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, Feb. 28, 2017, pp. 1146–1155. DOI: 10.48550/ARXIV.1702.08887. arXiv: 1702.08887 [cs.AI]. [Online]. Available: https://proceedings.mlr.press/v70/foerster17b.html.

[205] J. Jeon, W. Kim, W. Jung, and Y. Sung, "MASER: Multi-agent reinforcement learning with subgoals generated from experience replay buffer," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, Jul. 2022, pp. 10 041–10 052. DOI: https://doi.org/10.48550/arXiv.2206.10607. [Online]. Available: https://proceedings.mlr.press/v162/jeon22a.html.

[206] C. Daniel, M. Viering, J. Metz, O. Kroemer, and J. Peters, "Active Reward Learning," Jul. 2014. DOI: 10.15607/RSS.2014.X.031. [Online]. Available: http://www.roboticsproceedings.org/rss10/p31.pdf.

[207] G. Konidaris and A. Barto, "Autonomous shaping: Knowledge transfer in reinforcement learning," ICML '06, 489–496, 2006. DOI: 10.1145/1143844.1143906. [Online]. Available: https://doi.org/10.1145/1143844.1143906.

[208] B. Marthi, "Automatic shaping and decomposition of reward functions," pp. 601–608, 2007. [Online]. Available: https://icml.cc/imls/conferences/2007/proceedings/papers/358.pdf.

[209] M. Ghavamzadeh, S. Mahadevan, and R. Makar, "Hierarchical multi-agent reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 2, pp. 197–229, Apr. 2006. DOI: 10.1007/s10458-006-7035-4.

[210] S. Ahilan and P. Dayan, *Feudal multi-agent hierarchies for cooperative reinforcement learning*, Jan. 24, 2019. DOI: 10.48550/ARXIV.1901.08492. arXiv: 1901.08492 [cs.MA].

[211] G. Tesauro, "Extending Q-learning to general adaptive multi-agent systems," 2003. [Online]. Available: http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2003_CN16.pdf.

[212] S. R. Sinclair, S. Banerjee, and C. L. Yu, "Adaptive Discretization for Episodic Reinforcement Learning in Metric Spaces," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–44, Dec. 2019. DOI: 10.1145/3366703. [Online]. Available: https://doi.org/10.1145%2F3366703.

[213] F. L. D. Silva, M. E. Taylor, and A. H. R. Costa, "Autonomously reusing knowledge in multiagent reinforcement learning," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2018, pp. 5487–5493. DOI: 10.24963/ijcai.2018/774. [Online]. Available: https://doi.org/10.24963/ijcai.2018/774.

[214] F. L. Da Silva and A. H. R. Costa, "A survey on transfer learning for multiagent reinforcement learning systems," *Journal of Artificial Intelligence Research*, vol. 64, no. 1, 645–703, Mar. 2019, ISSN: 1076-9757. DOI: 10.1613/jair.1.11396. [Online]. Available: https://doi.org/10.1613/jair.1.11396.

[215] D. Shi, J. Tong, Y. Liu, and W. Fan, "Knowledge reuse of multi-agent reinforcement learning in cooperative tasks," *Entropy*, vol. 24, no. 4, p. 470, Mar. 2022,

ISSN: 1099-4300. DOI: 10.3390/e24040470. [Online]. Available: http://dx.doi.org/10.3390/e24040470.

[216] P. Abbeel and A. Y. Ng, "Apprenticeship Learning via Inverse Reinforcement Learning," in *Proceedings of the Twenty-First International Conference on Machine Learning*, ser. ICML '04, New York, NY, USA: Association for Computing Machinery, 2004, p. 1, ISBN: 1581138385. DOI: 10.1145/1015330.1015430. [Online]. Available: https://doi.org/10.1145/1015330.1015430.

[217] H. M. Le, Y. Yue, P. Carr, and P. Lucey, "Coordinated multi-agent imitation learning," pp. 1995–2003, 2017. eprint: 1703.03121. [Online]. Available: https://arxiv.org/pdf/1703.03121.pdf.

[218] F. L. Da Silva, G. Warnell, A. H. R. Costa, and P. Stone, "Agents teaching agents: a survey on inter-agent transfer learning," *Autonomous Agents and Multi-Agent Systems*, vol. 34, no. 1, pp. 1–17, Dec. 2019. DOI: 10.1007/s10458-019-09430-0. [Online]. Available: https://doi.org/10.1007/s10458-019-09430-0.

[219] P. Vrancx, Y. M. De Hauwere, and A. Nowe, "Transfer Learning for Multi-agent Coordination," in *ICAART 2011 - Proceedings of the 3rd International Conference on Agents and Artificial Intelligence, Volume 2 - Agents, Rome, Italy, January 28-30, 2011*, J. Filipe and A. L. N. Fred, Eds., SciTePress, 2011, pp. 263–272. DOI: 10.5220/0003185602630272.

[220] S. Kamboj and K. S. Decker, "Organizational Self-Design in Semi-dynamic Environments," vol. 5, pp. 1228–1235, May 2007. DOI: https://dl.acm.org/doi/10.1145/1329125.1329370.

[221] R. Kota, N. Gibbins, and N. R. Jennings, "Decentralized Approaches for Self-Adaptation in Agent Organizations," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 7, no. 28, pp. 1–28, Apr. 2012. DOI: 10.1145/2168260.2168261.

[222] D. Ye, "Self-organisation in multi-agent systems: theory and applications," University of Wollongong Thesis Collection 1954-2016, Doctor of Philosophy thesis, School of Computer Science and Software Engineering, University of Wollongong, 2013. [Online]. Available: http://ro.uow.edu.au/theses/3734.

[223] M. Viroli, J. Beal, F. Damiani, and D. Pianini, "Efficient Engineering of Complex Self-Organising Systems by Self-Stabilising Fields," *International Conference on Self-Adaptive and Self-Organizing Systems, SASO*, vol. 2015-Octob, no. Figure 1, pp. 81–90, 2015. DOI: 10.1109/SASO.2015.16.

[224] C. Bernon, V. Chevrier, V. Hilaire, and P. Marrow, "Applications of Self-Organising Multi-Agent Systems: An Initial Framework for Comparison," *Informatica*, vol. 30, no. 1, pp. 73–82, Mar. 2019. DOI: 10.15388/INFORMATICA.2018.198. [Online]. Available: https://www.informatica.si/index.php/informatica/article/download/75/66.

[225] F. Gechter, V. Chevrier, and F. Charpillet, "Localizing and tracking targets with a reactive multi-agent system, Proceedings," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.*, IEEE Computer Society, vol. 1, Brand: Association for Computing Machinery, 2004, pp. 1490–1491, ISBN: 1-58113-864-4.

[226] G. Franck, C. Vincent, and C. Francois, "A reactive multi-agent system for localization and tracking in mobile robotics," in *16th IEEE International Conference on Tools with Artificial Intelligence*, IEEE Comput. Soc, pp. 431–435, ISBN:

0-7695-2236-X. DOI: 10.1109/ICTAI.2004.15. [Online]. Available: http://ieeexplore.ieee.org/document/1374219/.

[227] Wikipedia contributors, *Holon (philosophy)*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Holon_(philosophy).

[228] M. Abdoos, A. Esmaeili, and N. Mozayani, "Holonification of a Network of Agents Based on Graph Theory," pp. 379–388, 2012. DOI: 10.1007/978-3-642-30947-2_42. [Online]. Available: https://ai2-s2-pdfs.s3.amazonaws.com/a7c5/90a82f6f8ac1c118350e349274cf4eab3187.pdf.

[229] F. Pichler, "Modeling complex systems by multi-agent holarchies," in *Computer Aided Systems Theory - EUROCAST'99*, P. Kopacek, R. Moreno-Díaz, and F. Pichler, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 154–168, ISBN: 978-3-540-44931-7.

[230] M. Vinyals, J. A. Rodriguez-Aguilar, and J. Cerquides, "A Survey on Sensor Networks from a Multi-Agent perspective," *The Computer Journal*, vol. 54, no. 3, pp. 455–470, Feb. 2010. DOI: {10.1093/comjnl/bxq018}.

[231] L. M. L. Oliveira and J. J. P. C. Rodrigues, "Wireless sensor networks: A survey on environmental monitoring," *Journal of Communications*, vol. 6, no. 2, pp. 143–151, 2011. DOI: 10.4304/jcm.6.2.143-151. [Online]. Available: http://www.jocm.us/index.php?m=content&c=index&a=show&catid=54&id=202.

[232] V. Ramasamy, "Mobile wireless sensor networks: An overview," *Wireless Sensor Networks—Insights and Innovations*, 2017.

[233] S. A. Munir, B. Ren, W. Jiao, B. Wang, D. Xie, and J. Ma, "Mobile Wireless Sensor Network: Architecture and Enabling Technologies for Ubiquitous Computing," in *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, vol. 2, 2007, pp. 113–120. DOI: 10.1109/AINAW.2007.257.

[234] F. Arena, G. Pau, and A. Severino, "V2X Communications Applied to Safety of Pedestrians and Vehicles," *Journal of Sensor and Actuator Networks*, vol. 9, no. 1, p. 3, Dec. 2019, ISSN: 2224-2708. DOI: 10.3390/jsan9010003. [Online]. Available: https://www.mdpi.com/2224-2708/9/1/3.

[235] M. Alsabaan, K. Naik, and T. Khalifa, "Optimization of Fuel Cost and Emissions Using V2V Communications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1449–1461, Sep. 2013. DOI: 10.1109/TITS.2013.2262175.

[236] M. T. Nguyen, C. V. Nguyen, H. T. Do, *et al.*, "Uav-assisted data collection in wireless sensor networks: A comprehensive survey," *Electronics (Switzerland)*, vol. 10, no. 21, pp. 1–24, 2021. DOI: 10.3390/electronics10212603.

[237] M. Pechoucek, "Defence Industry Applications of Autonomous Agents and Multi-Agent Systems," *Defence Industry Applications of Autonomous Agents and Multi-Agent Systems*, 2008. DOI: 10.1007/978-3-7643-8571-2.

[238] D. Baldazo, J. Parras, and S. Zazo, "Decentralized multi-agent deep reinforcement learning in swarms of drones for flood monitoring," *European Signal Processing Conference*, vol. 2019-September, Sep. 2019. DOI: 10.23919/EUSIPCO.2019.8903068.

[239] T. V. Safonova, O. N. Kolbina, N. V. Yagotintceva, and A. V. Mokryak, "The use of multi-agent systems in forestry," *IOP Conference Series: Earth and Environmental Science*, vol. 806, no. 1, 2021. DOI: 10.1088/1755-1315/806/1/012028.

[240] J. Scherer, S. Yahyanejad, S. Hayat, *et al.*, "An autonomous multi-UAV system for search and rescue," *DroNet 2015 - Proceedings of the 2015 Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, pp. 33–38, 2015. DOI: 10.1145/2750675.2750683.

[241] H. Wu, J. Wang, R. R. Ananta, V. R. Kommareddy, R. Wang, and P. Mohapatra, "Prediction based opportunistic routing for maritime search and rescue wireless sensor network," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 56–64, 2018. DOI: 10.1016/j.jpdc.2017.06.021. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2017.06.021.

[242] G. Xu, W. Shen, and X. Wang, "Marine environment monitoring using wireless sensor networks: A systematic review," *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, vol. 2014-Janua, no. January, pp. 13–18, 2014. DOI: 10.1109/SMC.2014.6973877.

[243] A. Gomez, M. F. Lagadec, M. Magno, and L. Benini, "Self-powered wireless sensor nodes for monitoring radioactivity in contaminated areas using unmanned aerial vehicles," *SAS 2015 - 2015 IEEE Sensors Applications Symposium, Proceedings*, Apr. 2015. DOI: 10.1109/SAS.2015.7133627. [Online]. Available: https://pub.tik.ee.ethz.ch/people/gomeza/GLMB2015_paper.pdf.

[244] P. Fang, Y. Z. Zhou, and Z. Xin, "A Remote Wireless Sensor Networks for Water Quality Monitoring," *CICC-ITOE 2010 - 2010 International Conference on Innovative Computing and Communication, 2010 Asia-Pacific Conference on Information Technology and Ocean Engineering*, pp. 364–365, 2010. DOI: 10.1109/CICC-ITOE.2010.9.

[245] M. Castillo-Effer, D. H. Quintela, W. Moreno, R. Jordan, and W. Westhoff, "Wireless sensor networks for flash-flood alerting," in *Proceedings of the Fifth IEEE International Caracas Conference on Devices, Circuits and Systems, 2004.*, vol. 1, IEEE, 2004, pp. 142–146. DOI: 10.1109/ICCDCS.2004.1393370.

[246] G. Werner-Allen, K. Lorincz, M. Welsh, *et al.*, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, 2006. DOI: 10.1109/MIC.2006.26.

[247] K. Goel and A. K. Bindal, "Wireless Sensor Network in Precision Agriculture: A Survey Report," in *2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, IEEE, Dec. 2018, pp. 176–181. DOI: 10.1109/PDGC.2018.8745854.

[248] M. P. Đurišić, Z. Tafa, G. Dimić, and V. Milutinović, "A survey of military applications of wireless sensor networks," in *2012 Mediterranean Conference on Embedded Computing (MECO)*, 2012, pp. 196–199, ISBN: 978-1-4673-2366-6.

[249] R. Min, M. Bhardwaj, S.-H. Cho, *et al.*, "Low-power wireless sensor networks," in *VLSI Design 2001. Fourteenth International Conference on VLSI Design*, 2001, pp. 205–210. DOI: 10.1109/ICVD.2001.902661.

[250] M. Prauzek, J. Konecny, M. Borova, K. Janosova, J. Hlavica, and P. Musilek, "Energy harvesting sources, storage devices and system topologies for environmental wireless sensor networks: A review," *Sensors (Switzerland)*, vol. 18, no. 8, 2018. DOI: 10.3390/s18082446.

[251] B. A. Warneke and K. S. J. Pister, "MEMS for distributed wireless sensor networks," in *9th International Conference on Electronics, Circuits and Systems*, vol. 1, 2002, pp. 291–294. DOI: 10.1109/ICECS.2002.1045391.

[252] D. Kandris, C. Nakas, D. Vomvas, and G. Koulouras, "Applications of wireless sensor networks: An up-to-date survey," *Applied System Innovation*, vol. 3, no. 1, pp. 1–24, Feb. 2020. DOI: 10.3390/asi3010014.

[253] J. Ebenezer and S. A. V. S. Murty, "Deployment of Wireless Sensor Network for radiation monitoring," *2015 International Conference on Computing and Network Communications, CoCoNet 2015*, pp. 27–32, Dec. 2015. DOI: 10.1109/CoCoNet.2015.7411163.

[254] D. Jha, S. Dahal, S. Shukla, B. Shahi, and P. G. Student, "Radioactive Contamination Detection in Water Using Wireless Sensor Network," *Internation Journal Of Advance Research And Innovative Ideas In Education*, vol. 1, no. 5, pp. 2395–4396, 2016. [Online]. Available: https://ijariie.com/AdminUploadPdf/radioactive_contamination_detection_in_water_using_wireless_sensor_network_1407.pdf.

[255] E. Felemban, F. K. Shaikh, U. M. Qureshi, A. A. Sheikh, and S. B. Qaisar, "Underwater Sensor Network Applications: A Comprehensive Survey," *International Journal of Distributed Sensor Networks*, vol. 2015, no. 11, p. 896 832, Nov. 2015. DOI: 10.1155/2015/896832.

[256] C. A. Pérez, M. Jimenéz, F. Soto, R. Torres, J. A. López, and A. Iborra, "A system for monitoring marine environments based on wireless sensor networks," *OCEANS 2011 IEEE - Spain*, pp. 0–5, 2011. DOI: 10.1109/Oceans-Spain.2011.6003584.

[257] T. Voigt, F. Osterlind, N. Finne, *et al.*, "Sensor Networking in Aquatic Environments - Experiences and New Challenges," in *32nd IEEE Conference on Local Computer Networks (LCN 2007)*, 2007, pp. 793–798. DOI: 10.1109/LCN.2007.23.

[258] C. Alippi, R. Camplani, C. Galperti, and M. Roveri, "A robust, adaptive, solar-powered WSN framework for aquatic environmental monitoring," *IEEE Sensors Journal*, vol. 11, no. 1, pp. 45–55, Jan. 2011. DOI: 10.1109/JSEN.2010.2051539.

[259] J. Lloret, M. Garcia, S. Sendra, and G. Lloret, "An underwater wireless group-based sensor network for marine fish farms sustainability monitoring," *Telecommunication Systems*, vol. 60, no. 1, pp. 67–84, 2015. DOI: 10.1007/s11235-014-9922-3.

[260] A. E. Adams, "ACMENet: an underwater acoustic sensor network protocol for real-time environmental monitoring in coastal areas," *IEE Proceedings - Radar, Sonar and Navigation*, vol. 153, no. 4, pp. 365–380, Aug. 2006, ISSN: 1350-2395. DOI: 10.1049/ip-rsn:20045060. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/ip-rsn_20045060.

[261] M. Kouzehgar, M. Meghjani, and R. Bouffanais, "Multi-Agent Reinforcement Learning for Dynamic Ocean Monitoring by a Swarm of Buoys," *in Proceedings of Global Oceans 2020: Singapore-US Gulf Coast (pp. 1-8). IEEE*, Dec. 21, 2020. DOI: 10.1109/ieeeconf38699.2020.9389128. arXiv: 2012.11641 [cs.RO]. [Online]. Available: http://arxiv.org/abs/2012.11641.

[262] E. Delory, A. Castro, C. Waldmann, *et al.*, "Objectives of the NeXOS project in developing next generation ocean sensor systems for a more cost-efficient assessment of ocean waters and ecosystems, and fisheries management," *Oceans 2014 - Taipei*, Apr. 2014. DOI: 10.1109/OCEANS-TAIPEI.2014.6964574.

[263] J. Pearlman, F. Pearlman, O. Ferdinand, *et al.*, "NeXOS, developing and evaluating a new generation of in-situ ocean observation systems," *OCEANS 2017 - Aberdeen*, vol. 2017-October, pp. 1–10, 2017. DOI: 10.1109/OCEANSE.2017.8084919.

[264] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, May 2009. DOI: 10.1016/j.adhoc.2008.06.003.

[265] S. S. Sonavane, V. Kumar, and B. P. Patil, "Designing wireless sensor network with low cost and low power," in *2008 16th IEEE International Conference on Networks*, 2008, pp. 1–5. DOI: 10.1109/ICON.2008.4772585.

[266] F. Wu, C. W. Tan, M. Sarvi, C. Rudiger, and M. R. Yuce, "Design and Implementation of a Low-Power Wireless Sensor Network Platform Based on XBee," in *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, 2017, pp. 1–5. DOI: 10.1109/VTCSpring.2017.8108667.

[267] M. Liu, J. Cao, G. Chen, and X. Wang, "An Energy-Aware Routing Protocol in Wireless Sensor Networks," *Sensors*, vol. 9, no. 1, pp. 445–462, 2009, ISSN: 1424-8220. DOI: 10.3390/s90100445. [Online]. Available: https://www.mdpi.com/1424-8220/9/1/445.

[268] S. Bhandari, N. Bergmann, R. Jurdak, and B. Kusy, "Time Series Data Analysis of Wireless Sensor Network Measurements of Temperature," *Sensors*, vol. 17, no. 6, p. 1221, May 2017, ISSN: 1424-8220. DOI: 10.3390/s17061221. [Online]. Available: https://www.mdpi.com/1424-8220/17/6/1221.

[269] G. M. Dias, M. Nurchis, and B. Bellalta, "Adapting sampling interval of sensor networks using on-line reinforcement learning," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, IEEE, Dec. 2016, pp. 460–465. DOI: 10.1109/WF-IoT.2016.7845391.

[270] C. Qian and H. Qi, "Coverage Estimation in the Presence of Occlusions for Visual Sensor Networks," in *Distributed Computing in Sensor Systems*, S. E. Nikoletseas, B. S. Chlebus, D. B. Johnson, and B. Krishnamachari, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 346–356, ISBN: 978-3-540-69170-9.

[271] L. Paradis and Q. Han, "A survey of fault management in wireless sensor networks," *Journal of Network and Systems Management*, vol. 15, no. 2, pp. 171–190, 2007. DOI: 10.1007/s10922-007-9062-0.

[272] N. H. Mak and W. K. G. Seah, "How long is the lifetime of a wireless sensor network?" *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, pp. 763–770, 2009. DOI: 10.1109/AINA.2009.138.

[273] A. A. Babayo, M. H. Anisi, and I. Ali, "A Review on energy management schemes in energy harvesting wireless sensor networks," *Renewable and Sustainable Energy Reviews*, vol. 76, pp. 1176–1184, Sep. 2017, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2017.03.124. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032117304598.

[274] F. Engmann, F. A. Katsriku, J. D. Abdulai, K. S. Adu-Manu, and F. K. Banaseka, "Prolonging the Lifetime of Wireless Sensor Networks: A Review of Current Techniques," *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–23, Aug. 2018. DOI: 10.1155/2018/8035065.

[275] J. B. Predd, S. R. Kulkarni, and H. V. Poor, *Distributed learning in wireless sensor networks*, Jul. 2006. DOI: 10.1109/MSP.2006.1657817. arXiv: 0503072 [cs].

[276] M. Gheisari, A. A. Abbasi, Z. Sayari, *et al.*, "A survey on clustering algorithms in wireless sensor networks: Challenges, research, and trends," in *2020 Interna-*

tional Computer Symposium (ICS), IEEE, Dec. 2020, pp. 294–299. DOI: 10.1109/ICS51289.2020.00065.

[277] M. Carlos-Mancilla, E. López-Mellado, and M. Siller, "Wireless Sensor Networks Formation: Approaches and Techniques," *Journal of Sensors*, vol. 2016, E. Llobet, Ed., p. 2 081 902, 2016, ISSN: 1687-725X. DOI: 10.1155/2016/2081902. [Online]. Available: https://doi.org/10.1155/2016/2081902.

[278] M. Mihaylov, Y.-A. L. Borgne, K. Tuyls, and A. Nowé, "Decentralised Reinforcement Learning for Energy-Efficient Scheduling in Wireless Sensor Networks," *Int. J. Commun. Netw. Distrib. Syst.*, vol. 9, no. 3/4, pp. 207–224, Aug. 2012, ISSN: 1754-3916. DOI: 10.1504/IJCNDS.2012.048871. [Online]. Available: https://doi.org/10.1504/IJCNDS.2012.048871.

[279] R. V. Kulkarni, A. Förster, and G. K. Venayagamoorthy, "Computational intelligence in wireless sensor networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 1, pp. 68–96, 2011. DOI: 10.1109/SURV.2011.040310.00002.

[280] A. Sharma and S. Chauhan, "A distributed reinforcement learning based sensor node scheduling algorithm for coverage and connectivity maintenance in wireless sensor network," *Wireless Networks*, vol. 26, no. 6, pp. 4411–4429, 2020. DOI: 10.1007/s11276-020-02350-y. [Online]. Available: https://doi.org/10.1007/s11276-020-02350-y.

[281] M. Iqbal, M. Naeem, A. Anpalagan, A. Ahmed, and M. Azam, "Wireless sensor network optimization: Multi-objective paradigm," *Sensors*, vol. 15, no. 7, pp. 17 572–17 620, Jul. 2015. DOI: 10.3390/s150717572.

[282] H. Li and Q. Zhang, "Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 284–302, 2009. DOI: 10.1109/TEVC.2008.925798.

[283] S. Sengupta, S. Das, M. D. Nasir, and B. K. Panigrahi, "Multi-objective node deployment in WSNs: In search of an optimal trade-off among coverage, lifetime, energy consumption, and connectivity," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 1, pp. 405–416, 2013, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2012.05.018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197612001248.

[284] P. N. Suganthan, "Differential evolution algorithm: Recent advances," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7505 LNCS, pp. 30–46, 2012. DOI: 10.1007/978-3-642-33860-1_4.

[285] M. Godzik, B. Grochal, J. Piekarz, M. Sieniawski, A. Byrski, and M. Kisiel-Dorohinicki, *Differential Evolution in Agent-Based Computing, 11th Asian Conference, ACIIDS 2019, Yogyakarta, Indonesia, April 8–11, 2019, Proceedings, Part II*. Springer International Publishing, 2019, vol. 11432 LNAI, pp. 228–241, ISBN: 9783030148010. DOI: 10.1007/978-3-030-14802-7_20. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-14802-7_20.

[286] K. R. Opara and J. Arabas, "Differential Evolution: A survey of theoretical analyses," *Swarm and Evolutionary Computation*, vol. 44, pp. 546–558, 2019, ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2018.06.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650217304224.

[287] M. Ayaz, A. Panwar, and M. Pant, "A Brief Review on Multi-objective Differential Evolution," in *Soft Computing: Theories and Applications*, M. Pant, T. K.

Sharma, O. P. Verma, R. Singla, and A. Sikander, Eds., Singapore: Springer Singapore, Feb. 24, 2020, pp. 1027–1040, ISBN: 978-981-15-0751-9. [Online]. Available: https://www.ebook.de/de/product/38676910/soft_computing_theories_and_applications.html.

[288] A. Čep and I. Fister, "Multi-agent System Based on Self-adaptive Differential Evolution for Solving Dynamic Optimization Problems," in *Proceedings of the 2017 4th Student Computer Science Research Conference*, University of Primorska Press, Oct. 2017, pp. 35–41, ISBN: 9789617023404. DOI: 10.26493/978-961-7023-40-4.35-41.

[289] Z. Zhang, Q. Han, Y. Li, Y. Wang, and Y. Shi, "An Evolutionary Multiagent Framework for Multiobjective Optimization," *Mathematical Problems in Engineering*, vol. 2020, 2020. DOI: 10.1155/2020/9147649.

[290] Y. Xu, J. Fang, W. Zhu, and W. Cui, "Differential Evolution for Lifetime Maximization of Heterogeneous Wireless Sensor Networks," *Mathematical Problems in Engineering*, vol. 2013, Y. Tang, Ed., p. 172 783, 2013, ISSN: 1024-123X. DOI: 10.1155/2013/172783. [Online]. Available: https://doi.org/10.1155/2013/172783.

[291] C. Lenzen and R. Wattenhofer, "Distributed algorithms for sensor networks," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 370, no. 1958, pp. 11–26, 2012.

[292] M. I. Khan and B. Rinner, "Performance Analysis of Resource-Aware Task Scheduling Methods in Wireless Sensor Networks," *International Journal of Distributed Sensor Networks*, vol. 10, no. 9, p. 765 182, Sep. 2014. DOI: 10.1155/2014/765182. [Online]. Available: https://doi.org/10.1155/2014/765182.

[293] M. Mihaylov, K. Tuyls, and A. Nowé, "Decentralized Learning in Wireless Sensor Networks," in *Adaptive and Learning Agents*, M. E. Taylor and K. Tuyls, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 60–73, ISBN: 978-3-642-11814-2.

[294] W. Guo, C. Yan, and T. Lu, "Optimizing the lifetime of wireless sensor networks via reinforcement-learning-based routing," *International Journal of Distributed Sensor Networks*, vol. 15, no. 2, p. 155 014 771 983 354, Feb. 2019. DOI: 10.1177/1550147719833541.

[295] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, IEEE, Oct. 1986, pp. 174–187. DOI: 10.1109/SFCS.1986.47.

[296] Y. Beck, I. Ljubić, and M. Schmidt, "A survey on bilevel optimization under uncertainty," *European Journal of Operational Research*, pp. 1–56, 2023. DOI: 10.1016/j.ejor.2023.01.008.

[297] B. Stadie, L. Zhang, and J. Ba, "Learning intrinsic rewards as a bi-level optimization problem," in *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, J. Peters and D. Sontag, Eds., ser. Proceedings of Machine Learning Research, vol. 124, PMLR, Aug. 2020, pp. 111–120. [Online]. Available: https://proceedings.mlr.press/v124/stadie20a.html.

[298] H. Zhang, W. Chen, Z. Huang, *et al.*, "Bi-level actor-critic for multi-agent coordination," *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, pp. 7325–7332, 2020, ISSN: 2159-5399. DOI: 10.1609/aaai.v34i05.6226. arXiv: 1909.03510.

[299] L. Canese, G. C. Cardarilli, L. Di Nunzio, *et al.*, "Multi-agent reinforcement learning: A review of challenges and applications," *Applied Sciences*, vol. 11, no. 11, p. 4948, May 2021. DOI: 10.3390/app11114948.

[300] M. van Otterlo and M. Wiering, "Reinforcement Learning and Markov Decision Processes BT - Reinforcement Learning: State-of-the-Art," M. Wiering and M. van Otterlo, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 3–42, ISBN: 978-3-642-27645-3. DOI: 10.1007/978-3-642-27645-3_1. [Online]. Available: https://doi.org/10.1007/978-3-642-27645-3_1.

[301] M. T. J. Spaan, "Partially Observable Markov Decision Processes BT - Reinforcement Learning: State-of-the-Art," in *Adaptation, Learning, and Optimization*, M. Wiering and M. van Otterlo, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 387–414, ISBN: 978-3-642-27645-3. DOI: 10.1007/978-3-642-27645-3_12. [Online]. Available: https://doi.org/10.1007/978-3-642-27645-3_12.

[302] L. N. Alegre, A. L. C. Bazzan, and B. C. da Silva, "Minimum-delay adaptation in non-stationary reinforcement learning via online high-confidence change-point detection," *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems. 2021. 97-105*, vol. 1, pp. 97–105, May 20, 2021. DOI: 10.48550/ARXIV.2105.09452. arXiv: 2105.09452 [cs.LG].

[303] S. Padakandla, K. J. Prabuchandran, and S. Bhatnagar, "Reinforcement Learning in Non-Stationary Environments," *Proceedings of the 5th ICACNI 2017*, vol. 708, pp. 23–31, 2020. arXiv: 1905.03970v4.

[304] S. Abdallah and M. Kaisers, "Addressing Environment Non-Stationarity by Repeating Q-Learning Updates," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1582–1612, Jan. 2016. DOI: {10.5555/2946645.2946691}.

[305] W. Mao, K. Zhang, R. Zhu, D. Simchi-Levi, and T. Basar, "Near-Optimal Model-Free Reinforcement Learning in Non-Stationary Episodic MDPs," *Proceedings of the 38th International Conference on Machine Learning*, vol. 139, pp. 7447–7458, 2021. [Online]. Available: https://proceedings.mlr.press/v139/mao21b.html.

[306] Wikipedia contributors, *Q-learning — Wikipedia, the free encyclopedia*, [Online; accessed 12-October-2022], 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Q-learning&oldid=1114881842.

[307] S. Steinarsson, "Downsampling Time Series for Visual Representation," M.S. thesis, University of Iceland, 2013, p. 87.

[308] J. Virkki, "Reliability of WSN Hardware," *International Journal of Embedded Systems and Applications*, vol. 1, no. 2, pp. 139–10, 2011. DOI: 10.5121/ijesa.2011.1201.

[309] H.-T. Ceong, H.-J. Kim, and J.-S. Park, "Discovery of and Recovery from Failure in a Costal Marine USN Service," *Journal of information and communication convergence engineering*, vol. 10, no. 1, pp. 11–20, Mar. 2012, ISSN: 2234-8255. DOI: 10.6109/jicce.2012.10.1.011.

[310] G. Xu, Y. Shi, X. Sun, and W. Shen, "Internet of things in marine environment monitoring: A review," *Sensors (Switzerland)*, vol. 19, no. 7, pp. 1–21, 2019. DOI: 10.3390/s19071711.

[311] P. Auer, T. Jaksch, and R. Ortner, "Near-optimal Regret Bounds for Reinforcement Learning," vol. 21, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., 2009, ISSN: 1532-4435. DOI: https://dl.acm.org/doi/10.5555/

1756006.1859902. [Online]. Available: https://dl.acm.org/doi/10.5555/1756006.1859902.

[312] O. Besbes, Y. Gur, and A. Zeevi, *Optimal Exploration-Exploitation in a Multi-Armed-Bandit Problem with Non-stationary Rewards*, May 13, 2014. DOI: 10.48550/ARXIV.1405.3316. arXiv: 1405.3316 [cs.LG].

[313] G. Xu, W. Shen, and X. Wang, "Applications of Wireless Sensor Networks in Marine Environment Monitoring: A Survey," *Sensors*, vol. 14, pp. 16 932–16 954, 2014, ISSN: 1424-8220. DOI: 10.3390/s140916932. [Online]. Available: https://www.mdpi.com/journal/sensors.

[314] S. Fattah, A. Gani, I. Ahmedy, M. Y. I. Idris, and I. A. T. Hashem, "A survey on underwater wireless sensor networks: Requirements, taxonomy, recent advances, and open research challenges," *Sensors*, vol. 20, no. 18, pp. 1–30, Sep. 2020. DOI: 10.3390/s20185393.

[315] A. Davis and H. Chang, "Underwater wireless sensor networks," in *2012 Oceans*, IEEE, Oct. 2012, pp. 1–5. DOI: 10.1109/OCEANS.2012.6405141.

[316] K. M. Awan, P. A. Shah, K. Iqbal, S. Gillani, W. Ahmad, and Y. Nam, "Underwater Wireless Sensor Networks: A Review of Recent Issues and Challenges," *Wireless Communications and Mobile Computing*, vol. 2019, p. 6 470 359, Jan. 2019, ISSN: 1530-8669. DOI: 10.1155/2019/6470359. [Online]. Available: https://doi.org/10.1155/2019/6470359.

[317] M. Jiang, Z. Guo, F. Hong, Y. Ma, and H. Luo, "OceanSense: A practical wireless sensor network on the surface of the sea," in *2009 IEEE International Conference on Pervasive Computing and Communications*, IEEE, Mar. 2009, pp. 1–5. DOI: 10.1109/PERCOM.2009.4912845.

[318] W. Elgenaidi and T. Newe, "Marine based Wireless Sensor Networks: Challenges and Requirements," *International Journal on Smart Sensing and Intelligent Systems*, vol. 7, no. 1178-5608, pp. 1–5, Jan. 2014. DOI: 10.21307/ijssis-2019-016.

[319] E. Liou, C. Kao, C. Chang, Y. Lin, and C. Huang, "Internet of underwater things: Challenges and routing protocols," in *2018 IEEE International Conference on Applied System Invention (ICASI)*, 2018, pp. 1171–1174. DOI: 10.1109/ICASI.2018.8394494.

[320] H. Lu, D. Wang, Y. Li, *et al.*, *CONet: A Cognitive Ocean Network*, 2019. arXiv: 1901.06253 [cs.CY].

[321] P. N. Mahalle, P. A. Shelar, G. R. Shinde, and N. Dey, "Introduction to Underwater Wireless Sensor Networks BT - The Underwater World for Digital Data Transmission," in P. N. Mahalle, P. A. Shelar, G. R. Shinde, and N. Dey, Eds., Singapore: Springer Singapore, 2021, pp. 1–21, ISBN: 978-981-16-1307-4. DOI: 10.1007/978-981-16-1307-4_1. [Online]. Available: https://doi.org/10.1007/978-981-16-1307-4_1.

[322] C. C. Kao, Y. S. Lin, G. D. Wu, and C. J. Huang, "A study of applications, challenges, and channel models on the Internet of Underwater Things," *Proceedings of the 2017 IEEE International Conference on Applied System Innovation: Applied System Innovation for Modern Technology, ICASI 2017*, no. 2, pp. 1375–1378, May 2017. DOI: 10.1109/ICASI.2017.7988162.

[323] M. Johansson and M. Sternad, "Resource allocation under uncertainty using the maximum entropy principle," *IEEE Transactions on Information Theory*, vol. 51, no. 12, pp. 4103–4117, Dec. 2005. DOI: 10.1109/TIT.2005.859277.

[324] P. Rygielski and J. M. Tomczak, "Context change detection for resource allocation in service-oriented systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6882 LNAI, no. PART 2, pp. 591–600, 2011. DOI: 10.1007/978-3-642-23863-5_60.

[325] J. Tomczak and O.-l. C. Detection, "On-Line Change Detection for Resource Allocation in Service-Oriented Systems Jakub Tomczak To cite this version : HAL Id : hal-01365566 On-Line Change Detection for Resource Allocation in Service-Oriented Systems," 2016.

[326] Sutton and R. Stuart, *Temporal credit assignment in reinforcement learning*, 1984. [Online]. Available: http://dl.acm.org/citation.cfm?id=911176.

[327] N. S. Pearre and H. Ribberink, "Review of research on v2x technologies, strategies, and operations," *Renewable and Sustainable Energy Reviews*, vol. 105, pp. 61–70, 2019, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2019.01.047. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032119300516.

[328] G. S. Oliveira, J. Roning, J. T. Carvalho, and P. D. M. Plentz, "Efficient Task Allocation in Smart Warehouses with Multi-Delivery Stations and Heterogeneous Robots," *2022 25th International Conference on Information Fusion, FUSION 2022*, no. i, 2022. DOI: 10.23919/FUSION49751.2022.9841337. arXiv: 2203.00119.

[329] K. H. Rosen, *Discrete Mathematics and Its Applications*. McGraw-Hill, 2018, p. 1120, ISBN: 9781260091991. [Online]. Available: https://faculty.ksu.edu.sa/sites/default/files/rosen_discrete_mathematics_and_its_applications_7th_edition.pdf.

[330] M. R. Ahmed, X. Huang, D. Sharma, and H. Cui, *Wireless Sensor Network: Characteristics and Architectures*, en, Dec. 2012. DOI: 10.5281/zenodo.1072589. [Online]. Available: https://doi.org/10.5281/zenodo.1072589.

[331] R. Dahiya, A. K. Arora, and V. R. Singh, "Modelling the Energy Efficient Sensor Nodes for Wireless Sensor Networks," *Journal of The Institution of Engineers (India): Series B*, vol. 96, no. 3, pp. 305–309, Aug. 2014, ISSN: 2250-2114. DOI: 10.1007/s40031-014-0149-1. [Online]. Available: https://doi.org/10.1007/s40031-014-0149-1.

[332] A. Kumar and S. Zilberstein, "Point-Based Backup for Decentralized POMDPs: Complexity and New Algorithms," in *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, 2010, pp. 1315–1322, ISBN: 978-0-9826571-1-9. [Online]. Available: http://rbr.cs.umass.edu/shlomo/papers/KZaamas10.html.

[333] A. R. Pinto, L. B. Poehls, C. Montez, and F. Vargas, "Power Optimization for Wireless Sensor Networks," in *Wireless Sensor Networks*, M. Matin, Ed., Rijeka: IntechOpen, 2012, ch. 2, pp. 23–50. DOI: 10.5772/50603. [Online]. Available: https://doi.org/10.5772/50603.

[334] M. A. Matin and M. M. Islam, "Overview of Wireless Sensor Network, Wireless Sensor Networks - Technology and Protocols," *IntechOpen*, 2012. DOI: 10.5772/49376. [Online]. Available: https://www.intechopen.com/chapters/38793.

[335] S. Escolar, S. Chessa, and J. Carretero, "Energy management in solar cells powered wireless sensor networks for quality of service optimization," *Personal and Ubiquitous Computing*, vol. 18, no. 2, pp. 449–464, Apr. 2013, ISSN: 1617-4917.

DOI: 10.1007/s00779-013-0663-1. [Online]. Available: https://doi.org/10.1007/s00779-013-0663-1.

[336] H. Sharma, A. Haque, and Z. A. Jaffery, "Modeling and Optimisation of a Solar Energy Harvesting System for Wireless Sensor Network Nodes," *Journal of Sensor and Actuator Networks*, vol. 7, no. 3, 2018, ISSN: 2224-2708. DOI: 10.3390/jsan7030040. [Online]. Available: https://www.mdpi.com/2224-2708/7/3/40.

[337] C. A. Trasviña-Moreno, R. Blasco, Á. Marco, R. Casas, and A. Trasviña-Castro, "Unmanned aerial vehicle based wireless sensor network for marine-coastal environment monitoring," *Sensors (Switzerland)*, vol. 17, no. 3, pp. 1–22, 2017.

[338] L. K. Ketshabetswe, A. M. Zungeru, M. Mangwala, J. M. Chuma, and B. Sigweni, "Communication protocols for wireless sensor networks: A survey and comparison," *Heliyon*, vol. 5, no. 5, e01591, May 2019. DOI: 10.1016/j.heliyon.2019.e01591. [Online]. Available: https://doi.org/10.1016/j.heliyon.2019.e01591.

[339] A. Kumar, V. Sharma, and D. Prasad, "Distributed Deployment Scheme for Homogeneous Distribution of Randomly Deployed Mobile Sensor Nodes in Wireless Sensor Network," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 4, 2013. DOI: 10.14569/IJACSA.2013.040422. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2013.040422.

[340] M. Bansal, I. Singh, and P. S. Sandhu, "Coverage and connectivity problem in sensor networks," en, *International Journal of Electronics and Communication Engineering*, vol. 5, no. 1, pp. 94–96, 2011, ISSN: 1307-6892. DOI: 10.5281/zenodo.1070907. [Online]. Available: https://publications.waset.org/vol/49.

[341] P. R. Pinheiro, Álvaro Meneses Sobreira Neto, and A. B. Aguiar, "Handing optimization energy consumption in heterogeneous wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 9, no. 9, p. 328 619, 2013. DOI: 10.1155/2013/328619. eprint: https://doi.org/10.1155/2013/328619. [Online]. Available: https://doi.org/10.1155/2013/328619.

[342] V. KumarSachan, S. Akhtar Imam, and M. T. Beg, "Energy-Efficient Communication Methods in Wireless Sensor Networks: A Critical Review," *International Journal of Computer Applications*, vol. 39, no. 17, pp. 35–48, Feb. 2012. DOI: 10.5120/4915-7484.

[343] M. A. Razzaque and S. Dobson, "Energy-efficient sensing in wireless sensor networks using compressed sensing," *Sensors (Switzerland)*, vol. 14, no. 2, pp. 2822–2859, 2014. DOI: 10.3390/s140202822.

[344] C. Avram, S. Folea, D. Radu, and A. Astilean, "Wireless radiation monitoring system," *Proceedings - 31st European Conference on Modelling and Simulation, ECMS 2017*, pp. 416–422, May 2017. DOI: 10.7148/2017. [Online]. Available: http://www.scs-europe.net/dlib/2017/ecms2017acceptedpapers/0416-mct_ECMS2017_0084.pdf.

[345] J. A. Boyan and M. L. Littman, "Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach," vol. 6, 1994.

[346] Y. Zhang and M. Fromherz, "Message-initiated constraint-based routing for wireless ad-hoc sensor networks," in *First IEEE Consumer Communications and Networking Conference, 2004. CCNC 2004.*, 2004, pp. 648–650. DOI: 10.1109/CCNC.2004.1286943.

[347] S. Lindsey and C. S. Raghavendra, “PEGASIS: Power-efficient gathering in sensor information systems,” *IEEE Aerospace Conference Proceedings*, vol. 3, pp. 1125–1130, 2002. DOI: 10.1109/AERO.2002.1035242.

[348] R. Arroyo-Valles, R. Alaiz-Rodriguez, A. Guerrero-Curieses, and J. Cid-Sueiro, “Q-Probabilistic Routing in Wireless Sensor Networks,” in *2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information*, IEEE, 2007, pp. 1–6. DOI: 10.1109/ISSNIP.2007.4496810.

[349] D. Marsh, R. Tynan, D. O’Kane, and G. M. P. O’Hare, “Autonomic wireless sensor networks,” *Engineering Applications of Artificial Intelligence*, vol. 17, no. 7, pp. 741–748, 2004. DOI: 10.1016/j.engappai.2004.08.038.

[350] D. Ye, M. Zhang, and Y. Yang, “A Multi-Agent Framework for Packet Routing in Wireless Sensor Networks,” *Sensors*, vol. 15, no. 5, pp. 10 026–10 047, Apr. 2015, ISSN: 1424-8220. DOI: 10.3390/s150510026. [Online]. Available: http://www.mdpi.com/1424-8220/15/5/10026/.

[351] A. Gupta, R. Singh, D. Ather, and R. S. Shukla, “Comparison of various routing algorithms for VANETS,” *Proceedings of the 5th International Conference on System Modeling and Advancement in Research Trends, SMART 2016*, pp. 153–157, 2016. DOI: 10.1109/SYSMART.2016.7894509.

[352] J. O. Gutierrez-Garcia and K. M. Sim, “Agent-based service composition in cloud computing,” *Communications in Computer and Information Science*, vol. 121 CCIS, pp. 1–10, 2010. DOI: 10.1007/978-3-642-17625-8_1.

[353] L. Qiu, “Self-Organization Mechanisms for Service Composition in Cloud Computing,” vol. 7, no. 2, pp. 321–330, 2014.

[354] F. Kiani, E. Amiri, M. Zamani, T. Khodadadi, and A. Abdul Manaf, “Efficient Intelligent Energy Routing Protocol in Wireless Sensor Networks,” *International Journal of Distributed Sensor Networks*, vol. 11, no. 3, p. 618 072, Mar. 2015, ISSN: 1550-1477. DOI: 10.1155/2015/618072. [Online]. Available: https://doi.org/10.1155/2015/618072.

[355] P. Wang and T. Wang, “Adaptive Routing for Sensor Networks using Reinforcement Learning,” *The Sixth IEEE International Conference on Computer and Information Technology (CIT’06)*, pp. 219–219, 2006. DOI: 10.1109/CIT.2006.34. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4019984.

[356] Z. A. Khan, O. A. Karim, S. Abbas, N. Javaid, Y. B. Zikria, and U. Tariq, “Q-learning based energy-efficient and void avoidance routing protocol for underwater acoustic sensor networks,” *Computer Networks*, vol. 197, p. 108 309, Oct. 2021, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2021.108309. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128621003212.