Distributed Task Allocation Optimisation Techniques in Multi-Agent Systems

Joanna Turner

Computer Science Department

Loughborough University

This dissertation is submitted for the degree of

Doctor of Philosophy

School of Science November 2018

Acknowledgements

There are many people I would like to thank for their support throughout my time as a PhD student. Firstly, I would like to thank my supervisor Dr. Qinggang Meng, who believed in my ability to do a PhD, and provided me with the opportunity to pursue a career in research. Most significantly, he helped me to develop my ability to think independently. I would also like to thank my second supervisor Dr. Gerald Schaefer, who offered valuable second opinions on important matters. I would like to thank Dr. Andrea Soltoggio who helped me to learn how to write scientifically, and regularly reminded me that the work I was doing was worthwhile. Thank you also to Dr. Amanda Whitbrook for her support, and for the opportunity to co-author a paper with her.

I would like to thank Dr. Shaheen Fatima, and Prof. Jon Timmis for agreeing and taking the time to be the examiners for my viva. Thank you to the Doctoral College, especially Dr. Duncan Stanley for the work that he does for PhD students, and for his valuable support and advice. Thank you also to Dr. Eugenie Hunsicker, Dr. David Sibley, Dr. Andy Archer, Dr. Ana Sălăgean, and Dr. Chris Hinde for their advice and support. I would like to thank the School of Science admin, finance, and IT teams for being so responsive and friendly while being under so much pressure. Thank you especially to Judith Poulton and Jo McOuat for being exceptionally helpful.

I would like to thank my colleagues, Dr. Richard Ellis-Braithwaite, Dr. Martins Irhebhude, Bob Nguyen, Carl Robinson, Angelika Skarysz, Mabelle Chen, Yanis Bahroun, and all my other office-mates throughout my time as a research student, who made these years such a unique experience. I would like to thank the School of Science lunchtime and monthly socials regulars. In particular, Dr. Daniel Reidenbach, Dr. Russell Lock, Prof. Ray Dawson,

Dr. Iain Phillips, Dr. Robert Mercaş, Dr. Paul Bell, Dr. Manfred Kufleitner, Dr. Dominik Freydenberger, and Dr. Amitabh Trehan who welcomed me into their lunchtime routine. Thank you very much to Dr. Christian Jäh, Dr. Konstantinos (Kostas) Kyriakopoulos, Dr Hideyasu Shimadzu (Shimadzu-Sensei), and to Dr. Yang Hu for being great company, and for being easy-going and inspiring people. I would also like to thank Dr. Firat Batmaz for always being so friendly and welcoming since as long as I have known him. Thank you Annette for being a great listener. Finally, I would like to thank my family who have been endlessly supportive throughout my PhD, and who also very generously allowed me to move back home during my writing-up phase.

Abstract

A multi-agent system consists of a number of agents, which may include software agents, robots, or even humans, in some application environment. Multi-robot systems are increasingly being employed to complete jobs and missions in various fields including search and rescue, space and underwater exploration, support in healthcare facilities, surveillance and target tracking, product manufacturing, pick-up and delivery, and logistics.

Multi-agent task allocation is a complex problem compounded by various constraints such as deadlines, agent capabilities, and communication delays. In high-stake real-time environments, such as rescue missions, it is difficult to predict in advance what the requirements of the mission will be, what resources will be available, and how to optimally employ such resources. Yet, a fast response and speedy execution are critical to the outcome.

This thesis proposes distributed optimisation techniques to tackle the following questions: how to maximise the number of assigned tasks in time restricted environments with limited resources; how to reach consensus on an execution plan across many agents, within a reasonable time-frame; and how to maintain robustness and optimality when factors change, e.g. the number of agents changes. Three novel approaches are proposed to address each of these questions. A novel algorithm is proposed to reassign tasks and free resources that allow the completion of more tasks. The introduction of a rank-based system for conflict resolution is shown to reduce the time for the agents to reach consensus while maintaining equal number of allocations. Finally, this thesis proposes an adaptive data-driven algorithm to learn optimal strategies from experience in different scenarios, and to enable individual agents to adapt their strategy during execution. A simulated rescue scenario is used to demonstrate the performance of the proposed methods compared with existing baseline methods.

Table of contents

List of figures

List of tables

List of Algorithms

Glossary

Agent An intelligent agent that independently carries out some set of operations using knowledge of an objective or goal.

CBBA The Consensus Based Bundle Algorithm is a distributed task allocation algorithm used as a benchmark in this thesis.

Conflict Used to refer to a conflicting assignment. This occurs when more than one agent is assigned to the same task..

Deadline The time at which a task must be started by an agent for it to be completed successfully.

Distributed system Networked agents communicate with each other and coordinate their actions using this communication.

EDF Earliest Deadline First.

Environment The observable context in which agents are situated.

NTF Nearest Task First.

Objective function A numerical quantity to be maximised or minimised.

PI The Performance Impact algorithm is a distributed task allocation algorithm used as a benchmark in this thesis.

Robot A physical autonomous agent capable of travelling to and completing tasks.

Task A job located in an environment to be completed by an agent.

Vehicle A physical autonomous agent capable of travelling to and completing tasks.

Chapter 1

Introduction

1.1 Motivation

In the field of computer science, an autonomous or intelligent agent is a computational entity that, through observations of its environment, independently makes decisions about how to act with the aim of achieving goals [104]. Examples of such agents include autonomous robots and software programs [101]. The use of robots has been investigated and applied in many real world settings such as manufacturing, search and rescue, space exploration, and in healthcare facilities. In such scenarios, robots offer benefits including the ability to work independently for an extended time in conditions that may be unsafe or unfavourable for humans. Similarly, software agents can automate complex or repetitive tasks [105].

Increasingly, multi-agent systems are being developed and deployed to accomplish more complex objectives with greater efficiency than a single agent is able to [90]. Applications include manufacturing [99, 30], logistics, and process control. These systems comprise of multiple agents, with possibly differing capabilities that solve different parts of the problem, working collaboratively to achieve goals that are broken down into several tasks. Multi-agent task allocation is an important research area as a wide range of real-world problems can be modelled as a multi-agent task allocation problem [105]. In a rescue operation, for example, autonomous drones equipped with cameras may be deployed to perform operations such as searching and monitoring survivors, and gathering data to analyse the structural integrity of

buildings. Meanwhile, ground teams may perform intricate rescue operations. Advances in computing and networking technology have enabled increasingly complex, large, distributed computing and information systems for complex applications, such as the internet, which would be infeasible for a centralised or single-agent system. With a greater number of collaborating agents comes a greater complexity for coordination, therefore designing an effective multi-agent system to ensure a successful outcome requires careful consideration of many factors:

- What is the application environment?
- What are the capabilities and limitations of the agents?
- How are tasks assigned to different agents?
- How does an agent prioritise and schedule its tasks?
- What information do agents have?
- How do agents communicate and synchronise their knowledge?
- Do agents have a common goal or individual goals?
- How are conflicting intentions and information handled?
- How are agents organised in terms of hierarchy?
- How does a change in the environment affect the behaviour of agents?

All of these factors and more can affect the success or optimality of the multi-agent team's performance in their environment. In this thesis, the questions of particular importance are of how tasks are assigned to different agents, how do agents prioritise and schedule their tasks, and how does changeability of factors such as the availability of agents affect the optimality of these strategies.

Equivalent task allocation problems exist in various and diverse application fields, for example the problem of assigning taxis to clients may be similar to the problem of assigning

cloud computing resources to users. Progress in solving the problem in one application may provide insight into solving similar problems in other applications. Agents and multi-agent systems therefore serve as a useful abstraction tool for the problem of task allocation.

1.2 Agent Definitions

Historically, the term 'agent' has attracted various definitions [34]. According to [101], an agent is generally understood as having the following essential characteristics: An agent is a computational entity i.e. a program that runs on a computational device. Agents are autonomous i.e. they have a certain degree of control over their own behaviour and can operate without human or other intervention. Agents are goal directed i.e. they perform tasks in such a way as to meet design objectives. An agent is said to be 'intelligent", which signifies that an agent pursues its goals such as to optimise defined performance metrics. An agent is said to be able to perform its tasks flexibly in diverse environments given the information it perceives through its sensors.

In a 'multi-agent system', two or more agents influence each others' goal driven behaviour through agent-to-agent interactions. Either through direct communication using a shared language, or indirectly by modifying the environment. Agents may have identical capabilities (homogeneous), such as a team of identical robots, or have different capabilities (heterogeneous), for example where one robot can dig, and another can fly. Agents may achieve coordination either through cooperation (the pursuit of the same goal), or through competition (the pursuit of conflicting goals), where competing agents aim to seek to achieve their own goals at the expense of others [101].

1.3 Distributed Systems

A multi-agent system has the features of a distributed system. A distributed system is one in which multiple computational nodes or agents are connected through a network and in which the agents work cooperatively to achieve objectives. In a distributed architecture, there is no

central controller or hierarchy, all agents are equal and autonomously make decisions [43]. The agents communicate only locally among neighbours and have no knowledge of the network topology [42]. Such systems are best suited for applications in which objectives can be broken down into multiple jobs or tasks, which can be autonomously performed by the different agents in the network. Examples of such systems include cloud computing systems, distributed sensor networks, and multi-robot teams. Figure 1.1 illustrates an example of distributed agents cooperating in a search and rescue mission. According to the literature, key desirable characteristics of a distributed system include:

- *mathematical soundness*, a predictable rate and a guarantee of convergence to a solution [70];
- *efficiency* by exploiting concurrent computation and avoiding the bottlenecks that exist with centralised controllers (provided that coordination overhead does not outweigh these benefits);
- *reliability* and *robustness*, the ability to react appropriately to failures, such as functioning agents compensating for the removal of an agent from the system due to damage;
- *scalability*, the system accommodates growing numbers of agents and tasks;
- *heterogeneity* agents with different capabilities collaborate to solve the problem;
- *responsiveness*, rapid adaptation to changes in the environment, such as the addition or removal of tasks [43].

1.4 Multi-Agent Task Allocation Problem

A task allocation problem aims to find a global feasible assignment of tasks to agents while optimising one or more objectives. As the numbers of tasks and agents grow, finding the optimal solution to a task allocation problem in real-time environments becomes computationally unfeasible. In complexity theory, the problem is said to be NP-hard [51].

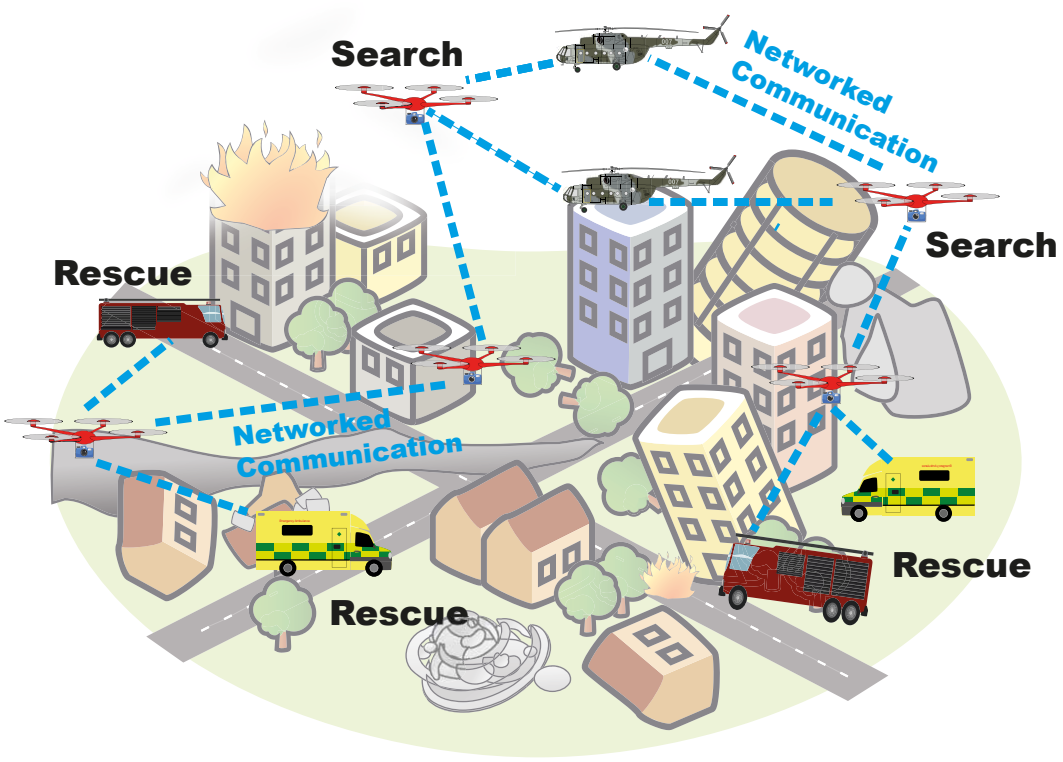


Fig. 1.1 An example of a search and rescue scenario involving multiple distributed communicating agents, cooperating to perform life saving tasks.

Constraints

The solution space of a task allocation problem may be reduced or made more complex by various constraints on how tasks can be assigned. The main constraints widely detailed in the literature are:

- Heterogeonity: Agents may have limited capabilities that restrict which types of tasks they can perform.
- Capacity constraints: such as limited fuel that restricts how many tasks agents can perform before needing to recharge.
- Time constraints: tasks may need to be completed within a certain time frame, such as delivering medical supplies in time to be useful.
- Complex tasks: some tasks may require more than one agent to be completed.

- Precedence constraints: such that some tasks may need to be performed before others. For example a blockage may need to be removed before supplies can be delivered to survivors in a rescue mission.

1.4.1 Utility

The process of determining the 'value' of a task to an agent with respect to the optimisation objective has previously widely been referred to as a utility function [51], a cost function, a score function [45], or an objective function [44]. Generally, the value of a task assignment is determined by:

- Cost: the cost of performing a task. For example, this can represent the fuel consumed to reach a task.
- Reward: this represents the gain of an agent performing a task, used as an incentive to perform that task. Rewards are often used in market-based approaches to task allocation.
- Utility: this can represent a combination of a reward and cost: Utility = Reward - Cost.

1.4.2 Optimisation Objectives

A task allocation problem may have one or more objectives to optimise. The optimisation objective may be to maximise some reward or score, or minimise a cost. These objectives can be equivalent. For example, the objective in a rescue mission may be to maximise the number of survivors rescued or to minimise the number of fatalities. Another objective could be to minimise the time taken to rescue all survivors. The accepted terminology for the objectives most commonly found in the literature is as follows. Here, an agent's path cost is equivalent to the sum of costs of all tasks in that agent's schedule:

- *MiniMax*: Minimise the maximum path cost over all the agents i.e. minimise the cost of the highest cost agent. For example. if the cost is duration time, the objective could

be to minimise the difference between the start of the first task, and the end of the last task.

- *MiniSum*: Minimise the sum of the path costs over all the agents. For example, if the cost is fuel consumption, the objective is to minimise the total fuel consumed by all the agents.
- *MiniAve*: Minimise the average task cost over all tasks. When the cost is waiting time, the objective is to minimise the average latency of tasks, between when the task is available and when the task is serviced.
- Maximise completed tasks, or minimise missed tasks.

1.4.3 Taxonomy

Gerkey and Matarić [36] devised a useful taxonomy to classify different task allocation problems for multi-robot task allocation (MRTA), which generalises to embodied agents. First, the taxonomy distinguishes between problems with single-task (ST) robots, and multi-task (MT) robots. ST robots can execute only one task at a time while MT robots can execute multiple tasks simultaneously. Second, the taxonomy distinguishes between problems with tasks that require only a single robot (SR) and tasks that require multiple robots (MR) to be completed. Third, the taxonomy distinguishes between problems in which robots have no planning capability, referred to as instantaneous assignment (IA), and in which robots can plan to execute multiple tasks according to a schedule, referred to as time-extended assignment (TA). Korsah et al. [51] introduced an extended version of this taxonomy, iTax, that covers the issues of interrelated utilities and constraints in task allocation problems. Three of the classifications are described:

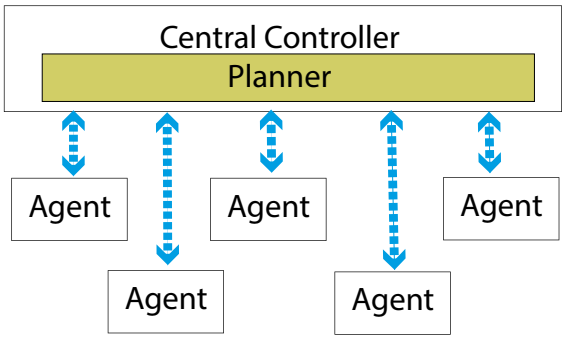
- No Dependencies (ND): the utility of an agent performing a task is independent of any other tasks or agents.

- In-schedule Dependencies (ID): the utility of an agent performing a task depends on the other tasks that the agent is planning to perform in its schedule. The agent can optimise its own schedule independently from other agents.
- Cross-schedule Dependencies (XD): the utility of an agent performing a task depends on the agent's own schedule and the schedules of other agents. The agent needs to consider the other agents' schedules when optimising its own schedule.

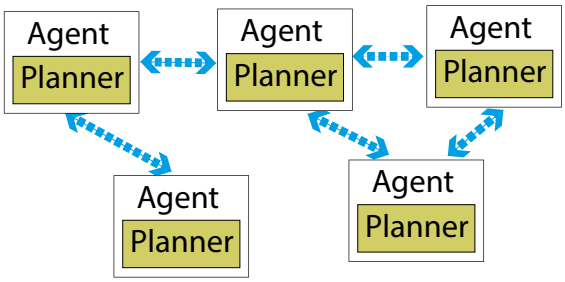
Nunes et al. [73] also extend Gerkey and Matarić's taxonomy by expanding the time-extended (TA) category to distinguish temporal and ordering constraints. Ordering constraints are expressed as synchronisation and precedence constraints (TA:SP). Temporal constraints are expressed as time windows (TA:TW), which impose lower and upper bounds on the start and end time of a task. This temporal constraint can be used to model relationships such as deadlines, which impose a constraint on the latest time an agent can arrive at a task before it expires. The authors distinguish between hard and soft temporal constraints. Hard temporal constraints can not be violated and are used in scenarios such as the delivery of perishable goods, and rescue missions. Soft temporal constraints can be violated but typically result in a penalty for the agent. The authors also distinguish between deterministic and stochastic models. With deterministic models, the initial conditions determine the output. With stochastic models, a degree of uncertainty is assumed and modelled.

1.4.4 Planning Architectures

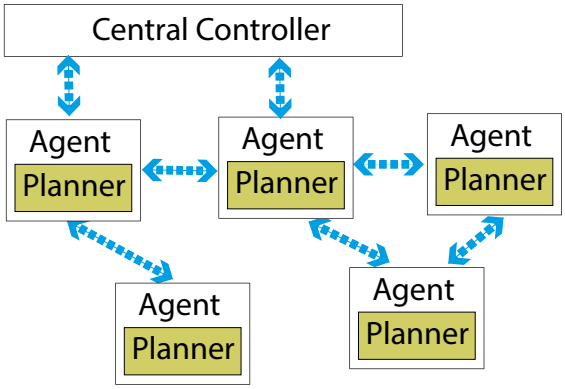
Multi-agent planning architectures can be classified under two main umbrellas: centralised and decentralised. Centralised architectures have the advantage of computing a global plan based on all available information, but have the main disadvantage of being a single point of failure. Decentralised and distributed planning avoids this pitfall by having the agents perform the planning. A hybrid architecture can exploit the benefits of having a centralised controller gather the information required for the mission, and exploit the robustness of distributed planning performed by the agents. Simple examples of these different types of architectures are illustrated in Figure 1.2.



(a) Centralised: A centralised planner gathers all necessary information, computes a plan for every agent in the team, and communicates the computed plans back to each agent. Interaction among agents is not required in the planning process.



(b) Distributed: Each agent is equipped with its own planner which computes a plan for that individual agent based on local information of the world. Agents then synthesise a global plan through interactions that facilitate conflict resolution.



(c) Hybrid: A distributed planning architecture that exploits centralised information gathering. A centralised server gathers information about the world from agents with sensing capabilities. The centralised server formalises a global list of tasks including the task properties, such as: location, deadline, and tools required to complete the task. The centralised server feeds this information to the distributed network of agents. Each agent is equipped with its own planner which computes a task schedule for that individual agent based on the information provided by the central server. Agents interact to synthesise a global plan.

Fig. 1.2 Multi-agent planning architectures. Dashed arrows symbolise two-way communication between agents and central controllers.

1.5 Multi-Agent Learning

The high complexity inherent in multi-agent optimisation problems presents a significant challenge in designing systems that generate optimal solutions, while also generalising to unseen environments. The field of multi-agent learning incorporates machine learning techniques to automate the optimisation process [76]. The rise of computational power of physical systems offers an increasingly attractive solution for multi-agent systems operating in real-time environments.

Machine learning is a fast growing domain that gives programs the ability to learn from and make predictions on data. It is a useful tool that enables agents to learn from experience to influence their decision making processes. Three broad categories of machine learning exist:

- Supervised learning: the learning algorithm uses labeled training data to infer a general rule or function that maps inputs to outputs. With classification supervised learning, the goal is for the learned function to correctly predict the classification of unseen instances.
- Unsupervised learning: the learning algorithm infers a function that describes the structure of data, where the data does not include a labeled classification.
- Reinforcement learning: An agent learns to optimise decision making within its environment through trial-and-error solely from a feedback of rewards and punishment.

A deeper investigation of multi-agent learning is outside the scope of this thesis. A comprehensive survey on cooperative multi-agent learning can be found Panait and Luke [76].

1.6 Complexity Theory

In computational complexity, algorithms are classified according to the resources required to run them as the inputs grow. The most commonly considered resources are time and storage. For task allocation problems, we are most interested in the running time i.e. the number

of operations or computational steps required by algorithms to solve the problem. Such a classification gives a useful indication of the scalability of algorithms, and of how algorithms designed to solve the same problem perform with relation to each other. We look at the worst case complexity, also known as big O notation, to determine the maximum time an algorithm will take to compute a solution as a function of the number of input variables. Inputs for the task allocation problem include the number of tasks and the number of agents.

Algorithms can be classified and compared generally according to whether they run in polynomial time, or exponential time. An algorithm is said to run in polynomial time if the number of computational steps required for it to complete, given an input n, is $\mathcal{O}(n^k)$, where k is a constant, non-negative integer. An algorithm runs in exponential time if the number of steps, given an input n, is $\mathcal{O}(k^n)$. Polynomial time algorithms are considered tractable and are therefore desirable in real-world environments. Algorithms in this class have a much slower growth rate, in general, compared to exponential time algorithms, which are generally considered to be intractable.

Classifications of problem complexity are used to determine the types of decision problems that can be solved quickly. The class P contains all decision problems that can be solved in polynomial time. Class NP (non-deterministic polynomial time) contains the set of decision problems for which solutions can be verified in polynomial time. The class NP-hard contains problems that are "at least as hard as the hardest problems in NP". Problems such as the Travelling Salesman problem and the task allocation problem investigated in this thesis belong to the NP-hard class of problems. There are no known polynomial algorithms that can solve an NP-hard problem optimally, therefore, heuristic or approximate methods have been developed to solve these problems in polynomial time.

1.7 The Task Allocation Problem of Interest

This thesis addresses the problem of distributed task allocation with time constraints for a networked team of agents. The task allocation problem considered requires that agents perform one task at a time, and each agent can be assigned multiple tasks that they execute

(a) Autonomous planning (b) Consensus

Fig. 1.3 Example of the task allocation process. Tasks 't' are located in a 2D space. Dashed arrows represent the path that an agent has chosen, for which their is a travel distance that agents must cover to reach the task. The dashed link between agents represents networked communication. Autonomous planning (a), agents independently determine which tasks to execute and in which order while respecting constraints and attempting to optimise a given objective. Consensus (b), agents communicate their task assignments among networked agents and resolve conflicting task assignments through a bidding process. The result is a conflict-free task allocation.

based on a schedule. The predicted cost of an agent performing a task depends on other tasks in that agent's schedule. Using the iTax taxonomy [51] and the expansion of Nunes et al. [73], this is known as the single-task (ST), single robot (SR), time-extended assignment (TA) with time windows (TW) problem with in-schedule dependencies (ID) i.e. The ID[ST-SR-TA:TW] class of task allocation problem. The task allocations are deterministic and the constraints are hard. Therefore, the scheduling of tasks can not violate any temporal constraints. Furthermore, a task can not be assigned to more than one agent, this is referred to as a 'conflict'.

Tasks are allocated to agents with the assumptions that agents autonomously decide which tasks to take on, and communicate with each other to reach consensus on which agents take which tasks. To determine which agents take which tasks, agents place bids on their selected tasks, share the bids by communicating with each other, and the agent with the highest bid (or equivalently lowest cost) wins the task. A simple example of this process is illustrated in Figure 1.3

1.8 Research Questions

Consider the task allocation problem described in Section 1.7. The research in this thesis aims to investigate the following research questions:

- In systems that employ heuristics to target optimisation objectives, a common issue occurs when the search reaches a local optimum. Given the objective of maximising the total number of allocated tasks, consider a scenario with a sub-optimal assignment of tasks when no more tasks can be directly added due to time constraints. Using local communications, can a sequence of task reassignments be coordinated that enables an increase in the number of allocated tasks, without unnecessarily disrupting the whole plan? Can the sequence of reassignments be initiated only when it will lead to an increase in allocations?

- In scenarios for which communication rounds are expensive, the time taken for the system to converge is an important factor. In distributed task allocation algorithms, can the time to reach consensus be reduced by removing variability in the consensus procedure? Can the quality of the solution be maintained by better exploiting scheduling heuristics?

- The task allocation problem is highly complex due to the high number and different combinations of variables, parameters and constraints. Accordingly, different heuristics perform better or worse under different such conditions. Using information derived from local communications, can each individual agent predict and adapt locally the best task allocation strategy to match the optimisation objective?

1.9 Key Contributions

This thesis addresses the problem of optimising distributed task allocation algorithms in scenarios where time constraints are critical. In particular, three extensions to existing state-of-the-art task allocation algorithms, PI [102] and CBBA [17], are proposed and analysed, providing optimisation techniques to maximise the number of task allocations, minimise

the time to reach a solution, and autonomously adapt to dynamic factors. The specific contributions of this thesis are as follows:

1. In scenarios with time constraints and a greater number of tasks than can be assigned, a key challenge is to find an allocation of tasks to agents that allocates the highest number of tasks possible. Due to the high complexity of the problem, distributed task allocation algorithms, such as PI and CBBA, use heuristic methods that generate sub-optimal solutions.

   Consider a scenario in which an agent A is the only agent capable of servicing a task t_1, due to time constraints. However, A's schedule is occupied by another task t_2. Consider another agent B that would be capable of servicing t_2, but not t_1. This scenario emerges because allocations are sought by heuristics that do not always find optimal solutions. In this situation, existing algorithms lack the ability to have the agents reassign t_2 to agent B so that agent A may service t_1. As a consequence, only one task is allocated when two could be allocated. As the problem size increases, so does the number of potentially assignable tasks left unassigned. A key limitation of algorithms such as CBBA, PI and other extensions of CBBA is therefore the inability of algorithms to reassign tasks after initial allocations had already been made, in order to fully exploit the time available in agents' schedules.

   The proposed solution, devised as part of the original contributions to this thesis, is to create feasible time slots for unallocated tasks. This contribution was published in [94, 97]. The principle idea is to ensure that an agent loses the bid for a task if a feasible slot could instead be created for an unassigned task, while also ensuring that the agent keeps the task if such a slot cannot be created. The proposed method requires agents to check whether a task, if removed from their schedule, would create a feasible slot for an unassigned task. If so, agents then place relatively low bids on those tasks according to the proposed bidding policy. Multiple reassignments among networked agents may be required to create a feasible time slot. The particular bidding policy

introduced allows for task reassignment chains that can involve a predefined maximum number of agents that can be adjusted according to performance requirements.

A simulated rescue scenario with task deadlines and fuel limits is used to demonstrate the performance of the proposed method compared with existing methods, the Consensus-Based Bundle Algorithm (CBBA) and the Performance Impact (PI) algorithm. Starting from existing solutions (PI-generated), results show an up to 20% increase in task allocations using the proposed method.

2. In highly dynamic and time critical environments, a fast convergence time is an essential property of a distributed algorithm. With consensus-based task allocation algorithms, the time it takes for all agents to converge is determined by the number of times the two phases of the algorithm repeat (as in Figure 1.3) until all agents reach consensus. This in turn is largely dependent on the number of conflicting task allocations i.e. when multiple agents bid for the same tasks.

   The second key contribution in this thesis, published in [96], is a proposed approach to reduce convergence time while maintaining the same or a higher number of task allocations. With previous methods, agents' bids on task assignments indicate the optimality of an assignment with respect to an optimisation objective. When conflicts occur, the agent that can perform the task most optimally keeps the assignment. For example, if the objective is to minimise the latency between a task becoming available and the time at which the task is serviced, bids may be based on agents' predicted arrival time at the task location. Changes in an agent's schedule are likely to affect its arrival times and therefore the bids placed on tasks. The proposed approach resolves conflicting task allocations based exclusively on agents' relative ranking in a hierarchy. Compared with using variable bids, the proposed rank-based approach stabilises the convergence process which has the effect of speeding up the rate of convergence.

   Certain insertion heuristics cause the CBBA to take a long time to converge but increase the number of allocated tasks. The proposed method enables the use of such heuristics while reducing the time to consensus. Two heuristics, earliest deadline first and

shortest travel time between tasks, are compared across different network topologies. Simulation results confirmed that the proposed rank-based conflict resolution approach was able to converge faster than the benchmark CBBA using variable bids. The findings suggested that the proposed approach is most effective and can significantly reduce the time to convergence when agents' ranks are determined by the network topology.

3. In consensus-based algorithms, a common approach in the autonomous scheduling phase is to have the agents determine which tasks to assign and in which order using a heuristic score function. The effectiveness of a given heuristic is dependent on various factors such as the problem constraints and the objective being optimised. Two well-known heuristics that perform well in time constrained scenarios are earliest deadline first (EDF) and nearest task first (NTF) [68]. The research presented in [95] proposes the idea that, given a choice of heuristics, agents can predict and select the best task inclusion heuristic locally, based on the limited information shared among networked agents. The proposed method extends CBBA with a learned prediction function in combination with a strategy switching behaviour. The method is effectively a prediction mechanism that uses past experience to select which task allocation strategy (i.e. heuristic) yields the optimal global task allocation. This method enables agents to independently adapt task allocation strategies in line with changing environmental factors, and thereby boost performance.

   To test the proposed method, the prediction function was trained to predict which heuristic, between EDF and NTF, yields the highest number of task allocations. Simulation results showed that for most scenarios tested, the agents were able to predict and select the optimal heuristic using locally communicated task assignment information. The method boosted performance in terms of the overall number of allocated tasks without a significant change in number of iterations until convergence.

1.10 Thesis Layout

The remainder of this thesis is organised as follows. Chapter 2 outlines the field of multi-agent ccordination and describes and discusses methods that have been previously developed, including centralised and distributed methods. The problem definitions for the contribution chapters are then described, along with detailed descriptions of the state-of-the-art distributed task allocation algorithms PI and CBBA. Chapter 3 introduces a proposed extension of PI, called PI-MaxAss, that increases the number of task allocations generated by PI. The proposed algorithm is formally described and a method to guarantee convergence is proposed for the PI algorithm as a whole. A complexity analysis of PI-MaxAss is also included. Finally, results are provided to demonstrate the performance of PI-MaxAss in a simulated search and rescue scenario. Chapter 4 introduces the fast consensus extension of CBBA. The proposed algorithm is described and its performance is then evaluated and compared with the baseline BW-CBBA. Chapter 5 Introduces the hybrid prediction and task selection model as an extension of CBBA. The algorithm and the training of the prediction function are described. The performance of the proposed method is then evaluated. Chapter 6 sumarises the contributions of this thesis and discusses possible future directions.

1.11 Publications Resulting from this Study

Conference Papers

- J. Turner, Q. Meng, and G. Schaefer, "Increasing allocated tasks with a time minimization algorithm for a search and rescue scenario." *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

- J. Turner, Q. Meng, and G. Schaefer, and A. Soltoggio "Fast Convergence for Fully Distributed Multi-Agent Task Allocation." *ACM/SIGAPP Symposium On Applied Computing (SAC)*, 2018.

- J. Turner, Q. Meng, and G. Schaefer, and A. Soltoggio "Distributed Strategy Adaptation with a Prediction Function in Multi-Agent Task Allocation." *17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2018.

Journal Paper

- J. Turner, Q. Meng, G. Schaefer, A. Whitbrook, and A. Soltoggio "Distributed Task Rescheduling With Time Constraints for the Optimisation of Total Task Allocations in a MultiRobot System." *IEEE Transactions on Cybernetics*, 2018.

Doctoral Consortium Extended Abstract

- J. Turner "Distributed Task Allocation Optimisation Techniques." *17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2018.

Chapter 2

Distributed Task Allocation: Definition and Current Approaches

This section presents existing approaches for multi-agent coordination, existing approaches for solving the task allocation problem, and the problem formulations for the task allocation scenarios tackled in the thesis.

2.1 Related Work

This section covers existing work related to the problem tackled in this thesis: related representations of the problem, general solution approaches, and state-of-the art distributed task allocation algorithms.

2.1.1 Similar Problems

The task assignment problem falls under the umbrella of combinatorial optimisation problems. In applied mathematics, combinatorial optimisation aims to find the best solution to a problem from a large set of possible solutions. Similarities can be found between the task assignment problem and other well-known problems in combinatorial optimisation, including the vehicle routing problem (VRP) [55], the travelling salesman problem (TSP) [33], and

other scheduling problems. In such problems, a brute-force approach that checks every possible solution is often impractical.

The VRP is a class of problems that aims to find the optimal set of routes for a group of vehicles to deliver goods to a set of customers. The vehicle routing problem with time windows (VRPTW) [50] is a variant of VRP that specifies time windows within which deliveries must be made. In vehicle routing problems, the assumption is that vehicles all start and end at the same depot, and that vehicles all have the same capabilities. VRP is a generalisation of TSP. A variant of the TSP that is similar to VRPTW is the Team Orienteering Problem with Time Windows (TOPTW) [98], also known as the Multiple Tour Maximum Collection Problem. The TOPTW considers multiple time-limited paths with the objective to maximise the total collected score over a set of vertices. This is comparable to multiple agents sequentially servicing tasks such as to maximise the total number of completed tasks. In TOPTW, each vertex is assigned a time window and is to be visited once at most. This is equivalent to the single task (ST) allocation problem with temporal constraints. A secondary objective to minimise the average latency (applied in Chapter 3) is comparable to another variant of the TSP, the K-Traveling Repairmen Problem (K-TRP) [26], also known as the Minimum Latency Problem. The K-TRP tries to determine a set of tours for multiple repairmen to visit a set of customers with the objective to minimise the average time a customer must wait before a repairman arrives. The multi-agent task allocation problem and TOPTW differ insofar as that the TOPTW specifies a start and end location between which the paths are created, whereas the task allocation problem may define a start point, but the final position is generally not specified.

2.1.2 Task Allocation Mechanisms

Two broad classifications for task allocation solutions in multi-agent systems have been covered extensively in the literature: centralised and decentralised task allocation [105]. Centralised task allocation systems, where a central server gathers information from each agent in the team and then computes an allocation for each agent, can optimise a chosen global objective based on a complete set of information from all agents. The drawbacks

are the resulting single point of failure, and the requirement that each agent must have a communication link with the central server. Thus, the possible mission range is limited, and a heavy communication and computation burden is put on the central server. Decentralised and distributed methods for task allocation overcome these limitations. In such cases, the task allocation algorithm runs on each agent simultaneously and the solution is reached through the interaction and exchange of information among them [25, 14, 48]. One of the drawbacks of distributed systems is that each agent has a different situational awareness and therefore communication and consensus procedures are required for the team of agents to reach agreement.

2.1.3 Centralised Task Allocation

While centralised task allocation is not the focus of this thesis, an overview of centralised approaches is included here for completeness.

Exact Solutions

Exact task allocation solutions are equivalent to optimal task allocation solutions. A brute-force approach that checks every possible solution is often impractical due to the high computation time and space required as the problem size increases. Branch-and-Bound is a centralised method that produces exact solutions more efficiently than an exhaustive search [22, 19, 58, 53]. The branch and bound algorithm represents the task allocation problem as a search tree. The algorithm explores branches of the tree, which represent candidate task allocation decisions, and keeps track of bounds on the solution that it is searching for. A branch is discarded if it cannot improve on the best solution found so far. Variants of Branch-and-Bound, which reduce the computational requirements of the algorithm by restricting the candidate solution search space, include Branch-and-Price [6, 84], Branch-and-Cut [5], and Branch-and-Price-and-Cut [63]. The latter has been used to solve the VRPTW [73, 4, 8]

Approximate Solutions

To generate solutions with a faster computation time than exact solutions, sub-optimal, but "good enough" approximate task allocations are sought through heuristic and nature-inspired approaches [75].

Various algorithms have explored strategies to solve multi-objective TSPs or vehicle routing problems, see [47] for a survey. Reference [77] tackles a bi-objective Traveling Salesman Problem with a two-phase local search procedure. The first phase generates a solution that optimises only one objective. The second phase begins the search from the solution generated in the first phase to optimise the second objective. The advantages to using this approach highlighted by [77] are to exploit the strong performance of single objective local search algorithms by chaining them together, and to maintain a flexible modularity and ease of understanding to the procedure that allows for modifications and enhancements. Heuristic methods to solve combinatiorial optimisation problems are prone to finding a local optimum [57]; however, a second search can perturb the first phase solution out of local optima to reach an enhanced solution closer to a non-dominated global optimum.

With larger multi agent teams, centralised solutions become intractable. Therefore decentralised methods are used to solve the multi-agent task allocation problem.

2.1.4 Decentralised Task Allocation

In this section, the focus is on decentralised approaches to multi-agent coordination. Several different branches of decentralised coordination have been developed that suit different applications with different priorities. The main approaches to decentralised task allocation stem from constraint optimisation, sequential decision-making, bio-inspired, and most relevant in this thesis, market-based.

Distributed Constraint Optimisation Problem (DCOP) approach

In constraint optimisation [21], problems are modelled as a set of constraints on variables, and the objective function is to be optimised in the presence of those constraints. The distributed

version of constraint optimisation is distributed constraint optimisation (DCOP) [32]. In DCOP, a group of distributed agents, each controlling some variables, select values in an attempt to minimise the cost of a set of constraints over those variables. The agents negotiate a solution through localised message exchanges. The authors in [15] provide a gentle tutorial on optimisation in multi-agent systems with a strong focus on DCOP techniques. The task allocation problem can be modelled as a DCOP [65, 73].

Complete algorithms have been developed i.e. algorithms that are guaranteed to find a configuration of variables that optimises the global objective function. The first complete DCOP algorithm that solved a special case of DCOP, called distributed constraint satisfaction, was the Asynchronous Backtracking Algorithm (ABT) [106]. The first DCOP algorithm proposed to solve the general problem optimally was ADOPT (Asynchronous Distributed OPTimization) [69]. With ADOPT, agents are arranged in a depth first search tree, and constrained agents need to be on the same branch. DCOP methods that are guaranteed to solve the problem optimally require an exponential communication overhead, large message sizes, or exponential computations performed by the agents. Distributed complete algorithms are therefore unsuitable for real-world applications [32].

Sub-optimal local search and inference-based approaches to solving DCOP have been developed to provide faster methods of computing good solutions. With local search techniques such as MGM (Maximum Gain Message) [64], agents perform local moves in parallel to optimise the local gain. The authors in [16] provide a unifying theory for local search techniques. Inference-based methods offer a different approach. Through iterative local message exchanges, each agent builds up an estimation of the impact that each of its actions has on the global objective. Once this estimation, also known as a belief function, is built up, each agent then selects the assignment that optimises the belief function. The max-sum approach is the most well-known of these approaches. Max-sum has been used to coordinate low-powered embedded devices [29], sensor-networks [28], and in Robocup Rescue [82, 78], however, it is also an algorithm with exponential complexity. In [81], the authors introduce Tractable High Order Potentials (THOP) to reduce Max-Sum's computational complexity

from exponential to polynomial time. The benchmarking platform RMASBench [49] was developed to compare the performance of different DCOP algorithms.

Sequential Decision-Making

Another widely studied framework for autonomous multi-agent planning, where planning refers to reasoning about which tasks will be implemented, is the Markov Decision Process (MDP) framework. MDPs, originally developed in operations research in the 1950s, provide a framework for modelling decision making in environments where there may be uncertainty about the state. The Markov property posits that the probability distribution for future states is based exclusively on the current state, and the action taken in the current state i.e. it is independent from previous states. Optimisation problems modelled as MDPs are often solved with dynamic programming and reinforcement learning [107].

A variant of MDP was developed to suit real world problems, in which information is incomplete, communication is costly, and it is unrealistic to have central decision-maker. Decentralised partially observable Markov decision process (Dec-POMDP) [2] is a variant of MDP that models control distributed across multiple agents with possibly differing and partial information about the environment. With Dec-POMDP, agents make choices based on local information, and the global reward depends on the actions taken by all the agents [2].

A key issue for MDPs is the high complexity for generating an optimal solution that quickly makes the methods intractable as the number of agents increases. It has been shown that the running time of Dec-POMDPs to find an optimal solution is NEXP-hard [7, 71]. More recent research has focused on the use of macro-actions in Dec-POMDPs, called MacDec-POMDP, that has increased the size of problems that can be solved practically [3]. With MacDec-POMDP, each robot temporally extended actions. These macro-actions allow for a higher level of abstraction, resulting in coordinated decisions occurring at a higher level.

When the environment is unknown or uncertain, and the models representing the environment are discrete and not too large, Dec-POMDP are well suited to solve discrete time sequential decision making planning problems under uncertainty. The MultiAgent Decision

Process (MADP) toolbox is a software platform for research in decision-theoretic multi-agent planning [86].

Bio-inspired

The principles of scalable self-organisation are found in natural biology, where large groups of insects or animals, such as ants, bees, and fish, collaborate without direct communication to form complex emergent global behaviour. In these systems, there is generally no global controller, and each individual acts based on local observations and a simple model of behaviour. This simplicity and localisation results in highly adaptive and scalable systems. These techniques are therefore well suited for scenarios in which distributed agents cannot communicate directly.

A self-organisation mechanism observed in social insects such as ants, in which coordination occurs through indirect interaction, is stigmergy [38]. Here, stigmergy refers to the sharing of information via modification of the environment. Ants leave pheromone trails that attract other ants, this mechanism facilitates optimal routes to food sources. These, and other bio-inspired self-organisation mechanisms have been used for coordination in fields such as robotics [52], routing protocols for mobile networks, and balancing workload of computers in a grid [27]. In [12], the authors propose a self-organisation method that allows a swarm of robots to assign tasks that have sequential interdependencies, using robots' perceptions of task delays.

The threshold-response model [11] is another bio-inspired mechanism that has been widely used for simple self organised division of labour. This model is based on the tendency of social insects to perform certain tasks. The authors in [31] use Swarm-GAP, an algorithm that adopts a probabilistic threshold model, for a distributed task allocation problem.

Contract-based

Contract-based methods have been developed, where two neighbouring agents or robots adjust their task allocation through mechanisms such as exchanges or task swaps [83]. Zheng and Koenig [109] developed a multi-robot distributed re-allocation mechanism called K-

swaps that describes multiple task exchanges among multiple agents at a time, and showed empirically that the method can optimise an existing task allocation solution by reducing team costs. Extending the idea of K-swaps, [62, 60, 61] introduced a decentralised task assignment algorithm considering instantaneous assignment, such that each robot is assigned exactly one task, the SR-ST-IA problem. The algorithm requires the differentiation of two roles, organiser and member robots, and can be used to optimise existing sub-optimal task assignments.

Market-Based Distributed Task Allocation

Market-based multi-robot coordination approaches [25] have been applied successfully to the ST-SR-TA problem to find sub-optimal solutions efficiently. With this approach, teams of self-interested agents iteratively trade tasks to maximise their own profit or minimise their costs. A cost is associated with an agent visiting a task within its path and is often measured as the total estimated use of individual resources to reach that task, such as fuel consumption, distance traveled or time to reach the target. The local cost of an agent's path is equal to the sum of costs of each task the agent is assigned to [54], and the global cost of an agent team is the sum of costs of all task assignments in the team. An auction is a commonly used market-based approach to assign tasks [72]. The process consists of several rounds of bidding in which agents place bids on each task where the value of a bid for a task is equal to the agent's estimated cost of visiting that task. The agent wins and is allocated those tasks for which it has placed a bid lower than any other agent. The effect of using this market-based approach is that local costs and subsequently global costs are minimised [25].

2.1.5 CBBA, Extensions and Variations

The Consensus-Based Bundle Algorithm (CBBA) [17] is a robust and fully distributed multi-assignment task allocation algorithm that employs a greedy auction strategy to enable agents to build a bundle of tasks sequentially. This task building phase is followed by a consensus procedure phase that resolves conflicting assignments. These two stages alternate until consensus has been reached by the team on all task assignments. For an analysis of CBBA's

scalability, see [17]. Of the various extensions and modifications, [18] and [41] address multi-robot (MR) task assignments and heterogeneous networks for the ST-MR-TA problem in which multiple robots may be required to service one task [36]. Choi et al. [18] address the case in which a task requires only one single agent, one or two agents, and exactly two agents of different type. Hunt et al. [41] propose the Consensus-Based Grouping Algorithm (CBGA) that addresses the problem of multi-agent multi-task assignment with group and equipment based dependencies, and which can accommodate any number of robots.

Ponda et al. [79] increase the overall efficiency of a task assignment by incorporating time windows of validity and fuel costs as part of the scoring scheme. The scoring scheme rewards agents for arriving at the optimal time for each task and for minimising fuel consumption. Ponda et al. [79] also address real-time re-planning for broken communication links, solving the problem of conflicting assignments when unconnected sub networks each have an agent assigned to the same task.

The Consensus phase of CBBA requires synchronised communication between all agents. In a real-time dynamic environment, coordinating a large number of agents to communicate in sync may overburden the network and require artificially delaying the broadcast of new messages until all earlier messages have been received by the network of agents. Johnson et al. [46] extend CBBA with an asynchronous communication protocol to permit the agents to run the consensus phase of the algorithm on their own schedule. The asynchronous communication protocol also uses less bandwidth than CBBA. In [80], the authors introduce CBBA with Relays algorithm that improves the team of agents' range and ensures network connectivity in a dynamic environment by utilising agents as communication relays.

Di Paola et al. [24, 23] propose the Heterogeneous Robots Consensus-based Allocation (HRCA) algorithm that deals with multi-assignments in heterogeneous networked-teams. The algorithm consists of two outer stages. Stage 1 iterates two inner phases that closely resemble the two phases of CBBA: As opposed to CBBA, in Stage 1 of HRCA the maximum task bundle size is ignored. Stage 2 is performed only if there exist bundles exceeding the maximum limit. In this case, iterative task elimination based on least penalty is performed to resize the bundle. Binetti et al. [9, 10] developed the Decentralised Assignment Algorithm

(DAA) based on CBBA and HRCA to solve the task allocation problem for assigning critical tasks for heterogeneous agents with limited capacity.

Cui et al. [20] introduce a Game Theory approach for task allocation. As with CBBA, the process of task allocation is split into two phases. A contract net protocol is used for the initial task allocation and a Game theory approach is then used to reallocate the tasks to satisfy Pareto Optimality. Smith et al. [85] extend CBBA to develop the Cluster-Formed Consensus-Based Bundle Algorithm (CF-CBBA) to reduce the communication necessary for reaching consensus on task allocation. The communication reduction has a trade-off of a drop in optimality of task allocation as complexity increases. The BW-CBBA [45] addresses the limitations of utilising DMG score functions to rank tasks within an agent's internal decision making process.

The authors in [108, 102] propose a concept called Performance Impact (PI) as an extension of CBBA. This method introduces PI, a value used by vehicles to prioritise task assignments. With PI, unlike CBBA, tasks included into a vehicle's task list can push back the execution times of later tasks in that same list, provided that all time constraints are satisfied. Likewise after a task is removed from a task list, the execution times of later tasks in the list may be shifted forward. With the PI algorithm, a vehicle does not release a task until it is reassigned elsewhere at a lower cost i.e. once a task is assigned it does not become unassigned. PI considers not only the cost of a task assignment but also the impact of that task assignment on the cost of other assignments in the vehicle's task list. The authors demonstrate the effectiveness of PI through a simulated rescue scenario with a global objective to minimise the average start times of tasks with deadlines. The PI algorithm was shown empirically to solve time-critical task allocation problems where CBBA could not, and was shown to find a lower average start time compared with CBBA. Despite the improved performance, the PI algorithm still fails to solve problems that are solvable due to converging to locally optimal but globally sub-optimal solutions [102].

2.1.6 Multi-Agent Learning: Related Work

Learning and adaptation in multi-agent systems is an established research field [100, 91, 88]. A commonly used approach to learning in multi-agent systems is reinforcement learning (MARL), in which agents learn actions and policies through trial and error from a feedback of rewards and punishment [56]. Extensive research has been done in this area. In early research, Littman [59] proposes a Markov games framework for MARL that allows for multiple adaptive agents with conflicting goals. Tan [92] investigates whether agents that learn cooperatively outperform agents that do not. The study showed that sharing learned policies could speed up learning with a cost in communication. Ho and Kamel [39] introduced a probabilistic hill-climbing approach to learning multi-agent coordination strategies that combines individual and group learning based on successful interactions in cooperative assignments. In more recent work, Garland and Alterman [35] developed distributed learning techniques to improve coordination among agents. By learning from past experiences of successful cooperation with other agents, and by learning probabilities of individual actions succeeding, agents were able to individually use past experience to more efficiently solve coordination problems. Empirical results demonstrated that distributed learning of individual agents improved performance of the whole system, including costs of communication and planning. Hu et al. [40] proposed knowledge transfer mechanisms to demonstrate how knowledge of individually learned policies can be utilised to learn better joint policies. The study exploited sparse interactions in multi-agent systems to improve the performance of multi-agent reinforcement learning. Marinescu et al. [66] introduced a predictive MARL approach that exploits prediction of future environment behaviour and pattern change detection capabilities to reduce the time needed for online learning. Panait and Luke [76] provide a a comprehensive survey of MARL, as well as evolutionary learning, for cooperative teams.

Reinforcement learning has recently been applied to applications with centralised task allocation architectures, such as cloud computing [74], where a scheduler that handles scheduling for multiple resources uses reinforcement learning to learn the best policies to reduce execution time. Gombolay et al. [37] proposed a method to automatically learn

scheduling heuristics from expert demonstrations using inverse reinforcement learning for a centralised scheduler.

2.2 Synthesis

With respect to the other approaches described, market-based approaches to task allocation have the major advantage of scalability. CBBA and its extensions are particularly suited to real-time applications due to CBBA's fully distributed design. All agents hold the same role in the task allocation process and therefore can be easily replaced if an agent is damaged. Agents communicate locally and use a bidding and consensus approach that guarantees convergence in polynomial time. The algorithm also has the advantage of a simple design. The two phases of the algorithm, path building and consensus, are intuitive to understand and easy to reproduce. These many beneficial reasons motivate the research in this thesis. In particular, the aim was to investigate how the two phases of this algorithm can be exploited and adapted to perform best under scenarios with different parameters, while maintaining the algorithm's performance guarantees. The research in this thesis investigates different bidding approaches, different path building approaches, how these perform together, and how agents can adapt these approaches to match environmental variables.

2.3 Consensus Based Bundle Algorithm (CBBA) and Performance Impact (PI) Overview

This section provides a description of CBBA, developed by [17], and PI, developed by [102], as well as their key features. A mathematical formulation of the algorithms will be introduced in a later section.

CBBA is a distributed auction-based task allocation algorithm for multi-agents and multi-tasks. The algorithm runs independently on each agent in a team of communication networked agents iterating over two phases: a bundle construction phase, and a task consensus phase. The algorithm guarantees convergence in polynomial time, even when agents have differences

in situational awareness, to a provably good approximate conflict-free task allocation solution. The algorithm therefore offers good scalability and is well suited to real-time environments.

2.3.1 Information Space

The local information space of each agent *i* is listed in Table 2.1. The information associated to each task *k* is listed in Table 2.2. Communication that an agent *i* sends to each neighbouring agent *j* is listed in table 2.3.

Table 2.1 local information space for each agent *i*

Agent *i*'s:	location
	travel speed
	fuel reserve
	capabilities for servicing certain types of tasks
	network connection to neighbouring agents
	local map of the environment
	local list of tasks that need to be serviced by the team
	local knowledge of winning bids and winning agents for all tasks
	local schedule (or bundle) of tasks that it plans to service

Table 2.2 Task information space

Task *k*'s:	location coordinates in the environment
	duration of service time
	deadline for starting the task
	task type

Table 2.3 Information communicated between agents

agent i sends to agent j:	agent i's local knowledge of winning bids for all tasks agent i's local knowledge of winning agents for all tasks

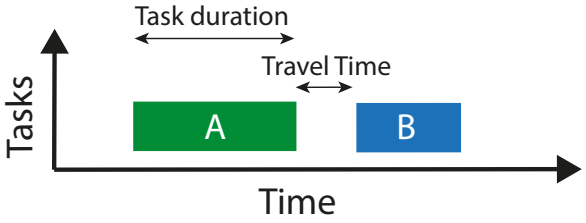


Fig. 2.1 Agent's schedule with two tasks A and B. With CBBA tasks added to the agent's schedule after A and B may not impact the start times of A and B. If A was added to the schedule before B and the agent loses the bid for task A, the agent releases both tasks A and B.

2.3.2 Bundle construction phase

In the bundle construction phase, agents incrementally and greedily build a local schedule by selecting one task at a time using a local score function. The task is selected from the full list of tasks for which the agent is capable of servicing. CBBA computes scores in terms of the improvement in overall bundle score as a result of adding the new task i.e. it is a marginal score. The task with the highest score, that is also higher than known bids from other agents, is selected as the next task to add to the schedule. When inserting the task into the schedule, the task needs to respect time constraints, and once added, newly added tasks cannot impact the start time of previously added tasks. This greedy task selection approach has the advantage of limiting the search to polynomial time. Figure 2.1 illustrates an agent's schedule where the length of tasks represents the task duration and the gap between tasks represents the travel time between tasks.

2.3.3 Consensus phase

In the consensus phase, agents communicate their local knowledge of winning agents and winning bids to their neighbours, and resolve conflicts according to a consensus protocol.

See [17] for details of the rules that determine who wins a bid. In general, the highest bid wins the task. If an agent is outbid for a task in its bundle, it releases that task as well as any tasks added subsequently to that task. The reason being that marginal scores computed for tasks added after the released task would no longer be accurate. In Figure 2.1, considering the case that the agent added task A and then added task B to its schedule, if the agent loses the bid for A, the agents removes both A and B from its schedule. The guarantee of convergence of CBBA requires that agents' score functions satisfy a property called Diminishing Marginal Gains (DMG). Informally, the condition requires that the score for a task cannot increase as more tasks are added to the bundle.

2.3.4 Performance Impact (PI)

The PI algorithm [102] extends CBBA with the following key modifications:

- PI minimises costs where CBBA maximises score.
- During the PI equivalent of the bundle construction phase, PI allows for the start times of tasks in a bundle to shift when a new task is added. This optimises the use of available time with a trade-off of increasing computation time. Starting from the schedule illustrated in Figure 2.1, with PI, an agent can insert a task C between task A and B by shifting the start time of B. This new schedule is illustrated in Figure 2.2.
- The next task included into an agent's schedule with PI is the task with the greatest positive difference between the current winning bid and the agent's score for that task i.e. the task with the greatest gain is selected.
- During the consensus phase, when an agent loses a bid, the agent releases only that task. Rather than releasing all tasks added later than the released task, as with CBBA. PI instead recomputes the start times of remaining tasks. In Figure 2.2, if an agent loses the bid for task A, with PI, the times for tasks C and B are recomputed to the earliest times the agent can reach them after task A is removed. The resulting schedule is illustrated in Figure 2.3. The advantage is to reduce the number of tasks to reassign, and

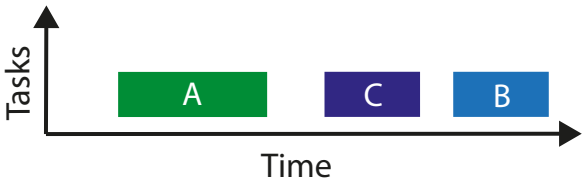


Fig. 2.2 Agent schedule with PI: task C can be inserted between tasks A and B by shifting the start time of task B.

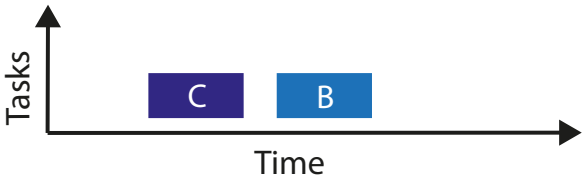


Fig. 2.3 Agent schedule with PI after losing the bid for task A. The start times for tasks C and B are recomputed.

optimise the start times of remaining tasks. The trade-off is an increase in computation time and an algorithm that does not satisfy the property of DMG. Therefore, cycling of assignments can occur between agents, and the algorithm then fails to converge.

2.4 Problem Formulation for PI algorithm

Introduced here are formal definitions for the task allocation problem addressed in Chapter 3, and formal definitions used to describe PI, as well as to describe the first novel contribution of this thesis, introduced as an extension of PI.

2.4.1 Multi-Vehicle Task Allocation

Consider a rescue scenario with n heterogeneous autonomous vehicles and m survivors. In this scenario, attending to a survivor is synonymous with executing a task. The goal is to provide targeted emergency support to the survivors as quickly as possible: e.g. some survivors may require food supplies, while others may require medical provisions. Thus in some scenarios different types of vehicles are necessary to complete different tasks. The

distributed vehicles in a network rely on local communication to co-ordinate a rescue plan over multiple iterations.

In the particular scenario considered, each survivor must be visited by one vehicle in order to be deemed rescued. Each vehicle can be assigned multiple targets and will sequentially visit those targets, while not required to return to its initial location. The main challenge is to reach an optimal allocation where allocation numbers are maximised and waiting time minimised, while respecting time constraints.

To formulate the problem mathematically, a set of n heterogeneous autonomous vehicles is defined by $\mathbf{V} = [v_1, \ldots, v_n]$, and a set of m tasks waiting to be completed is defined by $\mathbf{T} = [t_1, \ldots, t_m]$. A list of key symbols used hereafter is provided in Table 2.5. The ordered task allocation of the $i-th$ vehicle v_i is stored in $\mathbf{a}_i$, which can contain a variable number of tasks depending on how many tasks are assigned to v_i. Each task is to be assigned to one vehicle only, or left unassigned when time constraints cannot be satisfied.

Different task types can be executed by heterogeneous vehicles with the right capabilities. Thus, each task will be assigned only to vehicles functionally capable of performing them.

2.4.2 Task Assignment with Time Constraints

A latest start time s_k is defined for each task t_k after which it is too late for the task to be executed successfully; it is therefore necessary to determine whether a vehicle can arrive at the location of a task t_k before the latest start time s_k. The objective of minimising average waiting time measures the cost of a task assignment as the time it takes to start servicing the task from the start of the vehicle's schedule i.e. the total time the survivor must wait before being attended to. The time cost of a task t_k in $\mathbf{a}_i$, defined as $c_{i,k}(\mathbf{a}_i)$ in [108], is the predicted time taken by the vehicle v_i to arrive at the location of the task t_k. This time includes the duration of earlier tasks in $\mathbf{a}_i$ and travel time to and from those earlier tasks, but does not include the duration of the execution of t_k. For this particular scenario, the duration of a task is dependent on the task type [108]. Vehicles are additionally assumed to have limited fuel capacity that restricts the time a vehicle can be active for. All tasks must be started before the vehicle reaches its fuel capacity. The latest time at which v_i can arrive at a task before

reaching its fuel capacity is defined as f_i. The start time of the k-th task must therefore also be no later than f_i such that

$$c_{i,k}(\mathbf{a}_i) \leq min(s_k, f_i) \quad . \tag{2.1}$$

In [108] and [102], the global objective J is to minimise the average start time of all tasks, such that

$$J = \min \left\{ \frac{1}{m} \sum_{i=1}^{n} \sum_{k=1}^{|\mathbf{a}_i|} c_{i,k}(\mathbf{a}_i) \right\} \quad , \tag{2.2}$$

where $|\mathbf{a}_i|$ is the number of tasks assigned to v_i.

The objective function to maximise the number of allocated tasks is defined as

$$J^\star = \max \left\{ \sum_{i=1}^{n} |\mathbf{a}_i| \right\} \quad . \tag{2.3}$$

2.5 PI Algorithm

The PI algorithm is a distributed task allocation algorithm that runs simultaneously on each vehicle. Using the same two-phase architecture as CBBA, the PI algorithm iterates over a Task Inclusion phase and a Consensus and Conflict Resolution phase. During the first phase vehicles locally and iteratively build themselves a task bundle; during the second phase vehicles share their assignment lists with neighbouring vehicles and resolve conflicting assignments. Both phases repeatedly alternate until a global conflict-free task allocation is agreed upon by all vehicles. These main steps in an iteration of the algorithm are expressed with pseudocode in Algorithm 1.

The PI algorithm measures the local impact of a task assignment to the total cost of a vehicle's task list with the Removal Performance Impact (RPI) and the Inclusion Performance Impact (IPI) of a task assignment. The IPIs are computed during the Task Inclusion phase and determine which task to include next into a task list. The RPIs are computed at the end of the Task Inclusion phase and are communicated to networked vehicles during the Communication and Conflict Resolution phase. RPIs determine which vehicle keeps a task in case of conflict.

PI Task Inclusion Phase

The IPI of a task t_q in $\mathbf{a}_i$, as defined for PI-MinAvg, is measured as the time cost of t_q in $\mathbf{a}_i$ plus the sum of increase in time costs of other tasks in $\mathbf{a}_i$ that have been assigned previously. The increase in time costs occurs if later tasks need to be shifted to create enough time to service t_q. If no tasks have been assigned previously, the IPI of t_q in $\mathbf{a}_i$ is equal to its time cost, i.e. the time for v_i to reach t_q. This is because the sum of increase in time costs of other tasks in $\mathbf{a}_i$ is necessarily equal to 0. Let $\mathbf{a}_i \oplus_l t_q$ be the insertion of task t_q at position l in $\mathbf{a}_i$. The IPI of t_q in $\mathbf{a}_i$ is computed as

$$w_q^{\oplus}(\mathbf{a}_i, t_q) = \min_{l=1}^{|\mathbf{a}_i|+1} \{w_{q,l}^{\triangle}(\mathbf{a}_i, t_q)\} \quad , \tag{2.4}$$

where

$$w_{q,l}^{\triangle}(\mathbf{a}_i, t_q) = \sum_{z=l}^{|\mathbf{a}_i|+1} c_{i,z}(\mathbf{a}_i \oplus_l t_q) - \sum_{z=l}^{|\mathbf{a}_i|} c_{i,z}(\mathbf{a}_i) \quad . \tag{2.5}$$

Equation (2.5) computes the IPI of t_q at each position l in $\mathbf{a}_i$, where $c_{i,z}(\mathbf{a}_i)$ denotes the time cost of the task at position z in v_i's task list. Equation (2.4) finds the smallest IPI and records it as t_q's IPI in $\mathbf{a}_i$. A list to store the IPIs of each task is kept on each vehicle and is defined as $\gamma_i^{\oplus} = [w_1^{\oplus}, \ldots, w_m^{\oplus}]$ for vehicle v_i.

During this Task Inclusion phase, vehicles select tasks to include into their task lists until no more tasks can be added. This repeating process is depicted on lines 1–21 in Algorithm 2. Before including a task, the algorithm computes the IPIs of all candidate tasks t_q according to Equation (2.5) and (2.4), where candidate tasks are those compatible with v_i's capabilities

Algorithm 1 Task allocation outer-loop iterative procedure for CBBA and PI running on each vehicle

```
1: Initialise Timer T ← 1
2: converged ← false
3: while converged is false do
4:     Task Inclusion Phase
5:     Communication and Conflict Resolution Phase
6:     converged ← Check Convergence.
7:     T ← T + 1
8: end while
```

and not already in $\mathbf{a}_i$. The computation of IPIs is depicted on lines 3–12 in Algorithm 2. When there are already tasks in $\mathbf{a}_i$ that have been assigned previously it is necessary to determine which position in the task list yields the most optimal IPI, i.e. whether it is most optimal to include t_q at the start of $\mathbf{a}_i$, at the end, or in a position between tasks. Thus the IPI of t_q is computed in each position l (lines 5–9) and the position l in which the IPI is lowest is the optimal position (line 10).

After the IPIs of all candidate tasks have been computed, v_i selects for inclusion the task whose IPI can improve upon that task's current RPI the most. At this stage candidate tasks' RPIs will either have their initial value if unassigned, or an updated value received during the Communication and Conflict Resolution phase. RPIs for all tasks are initialised to their highest permissible cost such that RPIs of tasks must be lower than this value once they are assigned. An IPI of t_q in $\mathbf{a}_i$ lower than t_q's RPI in another vehicle's task list $\mathbf{a}_j$ indicates that

Algorithm 2 PI Task Inclusion phase

```
1:  while task list not full do
2:      w_q^⊕ ← highest permissible cost, w_q^⊕ ∈ γ_i^⊕
3:      for each task q do    // Compute IPI for each candidate task
4:          if task q is a candidate then
5:              for each insertion position l in task list do
6:                  if a_i ⊕_l t_q is feasible then    // If all time constraints are respected
7:                      Compute w_{q,l}^△ according to (2.5)    // Compute IPI in position l
8:                  end if
9:              end for
10:             Compute w_q^⊕ and position l according to (2.4)    // Keep minimum IPI
11:         end if
12:     end for
13:     Compute g from (2.6)    // Max difference between RPIs and IPIs for all tasks
14:     if g > 0 then
15:         Insert task q yielding g in position l of task list
16:         Update vehicle list β_q = i
17:         Update time costs of task list
18:     else
19:         break
20:     end if
21: end while
22: Compute γ_i (only RPIs in task list will be affected)
```

the global cost can be reduced if t_q is reallocated to v_i. The RPI of a task t_q is referred to formally as $w_q^{\ominus}$ and each vehicle stores the vector $\gamma_i = [w_1^{\ominus}, \ldots, w_m^{\ominus}]$. A task t_q assigned to v_j with an RPI greater than the IPI of t_q in $\mathbf{a}_i$ is written formally as $w_q^{\ominus}(\mathbf{a}_j, t_q) > w_q^{\oplus}(\mathbf{a}_i, t_q)$. Multiple IPIs may improve on the current RPIs, as such, v_i selects for inclusion the task that reduces the global cost most. The maximum difference between the RPIs of all tasks and the IPIs of all tasks is computed as:

$$g = \max_{q=1}^{m} \{\gamma_{i,q} - \gamma_{i,q}^{\oplus}\} \quad . \tag{2.6}$$

Line 13 in Algorithm 2 computes g according to (2.6). If $g > 0$ (line 14), the task corresponding to g is included into the vehicle's ordered task list, leading to the maximum reduction to the global cost. If $g \leqslant 0$, IPIs of all tasks are greater or equal to the current RPIs, meaning that the current assignments cannot be improved upon, or that time constraints of candidate tasks cannot be met. In this case the task inclusion process ends (line 19).

RPIs are updated at the end of the Task Inclusion phase (line 22). Whilst RPIs are constant for unassigned tasks, once assigned, the RPI is measured as t_k's time cost in $\mathbf{a}_i$ plus the sum of the changes in time cost of remaining tasks in $\mathbf{a}_i$ before and after the removal of t_k. By removing t_k from $\mathbf{a}_i$, v_i may be able to execute its remaining task assignments earlier. The time costs of tasks earlier in the task list than t_k are not affected by the removal of t_k. The RPI of a task t_k in $\mathbf{a}_i$ is formally

$$w_k^{\ominus}(\mathbf{a}_i, t_k) = \sum_{z=b}^{|\mathbf{a}_i|} c_{i,z}(\mathbf{a}_i) - \sum_{z=b+1}^{|\mathbf{a}_i|} c_{i,z}(\mathbf{a}_i \ominus t_k) \quad , \tag{2.7}$$

where b is the position of task t_k in v_i's task list, $c_{i,z}(\mathbf{a}_i)$ denotes the time cost of the task at position z in v_i's task list, and $\mathbf{a}_i \ominus t_k$ denotes $\mathbf{a}_i$ with t_k removed. When a global consensus is reached, all vehicles have an identical copy of γ.

PI Communication and Conflict Resolution Phase

Once the Task Inclusion phase is complete, the RPI list and an m-sized vehicle ID list that keeps track of which vehicle is assigned to which task, are broadcast to neighbouring vehicles.

The vehicle ID list is necessary for consensus and is defined as $\beta_i = [\beta_1, \ldots, \beta_m]$. Neighbouring vehicles are those where a communication link exists between two vehicles based on a network topology. This topology may be dynamic and depend on e.g. communication range and physical distance between two local vehicles. The vehicles communicate once per algorithmic iteration and the communication in this study does not consider a communication cost. As two or more vehicles may be assigned the same task, a consensus procedure introduced in [17] is used to resolve these conflicting assignments. A lower RPI indicates a more optimal assignment, therefore vehicles with a higher RPI for a conflicting assignment release the task. RPIs and associated vehicle IDs are updated during consensus.

The Task Inclusion and Conflict Resolution phases repeat until no inclusions or removals can be made. At this point, the system is deemed to have converged and the task allocation procedure ends.

2.6 Problem Formulation for BW-CBBA extensions

Introduced here are formal definitions for the task allocation problem addressed in Chapters 4 and 5, and formal definitions used to describe CBBA, its extension BW-CBBA, as well as to describe the second and third novel contributions of this thesis, introduced as extensions of CBBA. The contributions are a proposed approach to reduce the time to reach consensus, and a proposed approach to increase agents' ability to adapt their strategies such as to increase the optimality of the task allocation.

2.6.1 Basic Definitions

Given a team of n agents and m tasks, the problem of interest is to allocate tasks to agents with the following assumptions: agents autonomously decide which tasks to take on using a scoring function that computes a score for that agent to perform a certain task. These score functions often incorporate heuristics designed to optimise a specified objective. Agents then communicate with each other to reach consensus on which agents take which tasks. To determine which agents take which tasks, agents place bids on their selected tasks, share

the bids by communicating with each other, and the agent with the highest bid wins the task. Agents co-operate to maximise the number of allocated tasks and to reach an agreed allocation (consensus). Tasks and agents are subject to time constraints.

Formally, $\mathbf{V} = [v_1, \ldots, v_n]$ and $\mathbf{T} = [t_1, \ldots, t_m]$ represent the set of n agents and m tasks, respectively. Each agent $v_i \in \mathbf{V}$ is initialised with:

- A bundle $\mathbf{b}_i$ of tasks assigned to v_i ordered chronologically based on when the tasks were added. Newly assigned tasks are appended to the end of the bundle.
- A path $\mathbf{p}_i$, same as $\mathbf{b}_i$, but with tasks in the order in which v_i will execute them.

To select which tasks to add to the bundle, an agent computes a score c_{ik} for each task $t_k \in \mathbf{T}$ using a function $F_{ik}()$. Agents can take on up to L_t tasks. The length of the bundle and path, represented by $|\mathbf{b}_i|$ and $|\mathbf{p}_i|$ respectively, must be therefore less than or equal to L_t.

- A winning agent list $\mathbf{z}_i = [z_{i1}, \ldots, z_{im}]$ where an element z_{ik} stores the index of the agent who has won the task t_k according to the latest communication received by v_i. If v_i has not received or made a bid on t_k, then $z_{ik} = 0$.
- A winning bid list $\mathbf{y}_i = [y_{i1}, \ldots, y_{im}]$ where an element y_{ik} stores the winning bid for t_k corresponding to the winner z_{ik}. If there is no bid for task t_k, then $y_{ik} = 0$. Bids on tasks are greater than 0 and less than or equal to $MaxBid$.

2.6.2 Problem Constraints

Agents can perform at most one task at a time, and each agent can be assigned multiple tasks that they execute based on a schedule, with travel times between tasks. Each agent has a maximum operating time f_i, which is the latest time at which v_i can arrive at a task t_k before running out of fuel. Each task t_k has a latest start time ξ_k after which the task expires. The predicted time of execution of $t_k \in \mathbf{p}_i$ by v_i is ς_{ik}. This time includes the duration of earlier tasks in $\mathbf{p}_i$ and travel time to and from those earlier tasks. Thus,

$$\varsigma_{i,k} \leq min(\xi_k, f_i) \quad . \tag{2.8}$$

Due to these time constraints, it may not be possible to assign all tasks. If a task is not already in $\mathbf{p}_i$ and satisfies the time constraints, it is a *candidate task* and can be considered for inclusion.

Agents communicate with each other via links determined by a network topology. This topology may be restricted, e.g. by communication range. In dynamic settings, the topology may change and become disconnected when agents move [79]. In this study, the agents are stationary during the task allocation process, the topology remains the same and is connected. Once a plan has been agreed, the agents set off to perform their assigned tasks.

2.6.3 Objective Function

The primary global objective $J^\star$ for the problem of interest is to maximise the number of allocated tasks, formally defined as

$$J^\star = \max \left\{ \sum_{i=1}^{n} |\mathbf{p}_i| \right\} \tag{2.9}$$

$$\text{s.t.} \quad \mathbf{p}_i \cap \mathbf{p}_j = \emptyset, \text{where} \quad i \neq j \tag{2.10}$$

The constraint states that the tasks in $\mathbf{p}_i$ may not be in any other agents' paths i.e. a task may be assigned to one agent's task list at most.

A secondary global objective J is to minimise the average start time of all tasks, such that

$$J = \min \left\{ \frac{1}{m} \sum_{i=1}^{n} \sum_{k=1}^{|\mathbf{p}_i|} \varsigma_{i,k}(\mathbf{p}_i) \right\} \quad . \tag{2.11}$$

2.7 CBBA and BW-CBBA

In this section, CBBA and its extension BW-CBBA are formally defined.

CBBA

CBBA iterates over the following two phases:

1. The *bundle building phase*: each agent greedily builds up a bundle and path through a repeating process of computing scores for each candidate task and selecting the task with the highest score to add to their bundle and path.

2. The *consensus phase*: agents communicate $\mathbf{z}_i$ and $\mathbf{y}_i$ to neighbouring agents i.e. those with communication links based on a network topology. When there are conflicting assignments, the highest bid wins and losing agents remove the task from their bundles as well as all tasks that were added to the bundle after that task. If bids are tied then the agent with the lowest index wins the task. [18]

As the consensus phase results in agents removing tasks from their bundles and creating time in their schedules, the bundle building phase is repeated to attempt to assign more tasks in the free time that is available. The two phases alternate until agents can no longer add tasks into their schedules and consensus has been reached by the team on all task assignments, such that all agents have an identical list $\mathbf{z}_i$. CBBA converges in polynomial time, within $max\{m, L_t n\} \cdot D$ iterations where D is the diameter of the network, provided that the scoring function satisfies diminishing marginal gain (DMG) [17]. DMG means that the score that v_i computes for candidate task t_k, defined as c_{ik}, cannot increase as a result of other tasks being added to the bundle $\mathbf{b}_i$ before t_k [17], such that:

$$c_{ik}(\mathbf{b}_i) \leq c_{ik}(\mathbf{b}_i \oplus_{end} t_z) \quad , \tag{2.12}$$

where $\oplus_{end}\, t_z$ denotes the append of t_z to the end of $\mathbf{b}_i$. The trade off of the DMG condition is a possible performance degradation in certain scenarios.

Bid Warped CBBA

The Bid Warped CBBA (BW-CBBA) [45] decouples the *scores* that inform task selection in the bundle building phase (internal scores) from the *bids* that are communicated to networked

agents (external scores). The idea is that the internal score function need not satisfy DMG and the external bids need not be identical to the internal scores. Only the bids that agents share with each other need to satisfy DMG to guarantee convergence. A proof is provided in [45].

Score Functions

To determine the score of a candidate task t_q, CBBA inserts t_q into $\mathbf{p}_i$ at each index l one at a time. A constraint is that the insertion cannot impact the current start times for the tasks already in the path [79] and - for the implementation in this study - satisfies the time constraints in equation (2.8). The score is computed at each index l and the highest score is stored as c_{iq}. The score function is defined as:

$$F_{iq}(\mathbf{p}_i \oplus_l t_q) = R_{iql} - C_{iql}, \tag{2.13}$$

and

$$c_{iq} = \max_l F_{iq}(\mathbf{p}_i \oplus_l t_q), \tag{2.14}$$

where $\mathbf{p}_i \oplus_l t_q$ denotes the inclusion of t_q into $\mathbf{p}_i$ at index l. R_{iql} denotes the reward and C_{iql} the cost for including t_q into $\mathbf{p}_i$ at index l. If the insertion of t_q cannot meet the constraints at any index in the path, then $c_{iq} = 0$.

BW-CBBA applies a bid warping function to c_{iq} that produces a DMG satisfying score $\bar{c}_{iq}$. The bid warping function G is defined as:

$$\bar{c}_{iq} = G_{iq}(c_{iq}, \mathbf{b}_i) = \min(c_{iq}, \bar{c}_{iq_j}) \quad \forall j \in \{1, \ldots, |\mathbf{b}_i|\} \tag{2.15}$$

where $\bar{c}_{iq_j}$ is the score of the jth element in the current bundle [45]. In other words, the bid for a candidate task must be lower than, or is made to be equal to, the lowest bid of all other tasks already in agent v_i's bundle.

Bid Warped CBBA Bundle Building Phase

The bundle building phase of BW-CBBA [45] that runs independently on each agent v_i is summarised in Algorithm 3: For each candidate t_q, v_i computes a score c_{iq} with its internal score function F_{iq} (line 4). A DMG satisfying bid $\bar{c}_{iq}$ is then created with the function G_{iq} (line 5). $\bar{c}_{iq}$ is compared with the current winning bid y_{iq} for t_q. The boolean $h_{iq} = true$ if v_i outbids the current winner (line 6). The candidate task selected to be added to v_i's task list (using task index $q^\star$) is the task that has the highest score c_{iq} that also outbids the current winner (line 8). The new winning agent for $t_{q^\star}$ is set as v_i's index in z_i (line 10). The winning bid for $t_{q^\star}$ is set as $\bar{c}_{iq^\star}$ in y_i. Then, $t_{q^\star}$ is appended to the bundle, and inserted into the path where it yielded the highest score $c_{iq^\star}$. The bundle building phase terminates when no candidate tasks can outbid the current winning bids, or the maximum bundle length is reached.

Table 2.4 Symbol Definitions for PI algorithm

Symbol	Definition
$\mathbf{V} = [v_1, \ldots, v_n]$	Set of n vehicles
$\mathbf{T} = [t_1, \ldots, t_m]$	Set of m tasks
$\mathbf{a}_i$	Ordered task allocation of the i^{th} vehicle v_i
s_k	Latest start time for task t_k
$c_{i,k}(\mathbf{a}_i)$	The time cost of a task t_k in $\mathbf{a}_i$: the predicted time taken by v_i to arrive at the location of the task t_k in its schedule $\mathbf{a}_i$
f_i	The latest time at which v_i can start a task before running out of fuel
$\lvert\mathbf{a}_i\rvert$	The number of tasks assigned to v_i
$w_k^{\ominus}(\mathbf{a}_i, t_k)$	The Removal Performance Impact (RPI) of a task t_k in $\mathbf{a}_i$
$\mathbf{a}_i \ominus t_k$	$\mathbf{a}_i$ with t_k removed
$\gamma_i = [w_1^{\ominus}, \ldots, w_m^{\ominus}]$	Vector on each vehicle to store RPIs
$\mathbf{a}_i \oplus_l t_k$	The inclusion of task t_k at position l in $\mathbf{a}_i$
$w_q^{\oplus}(\mathbf{a}_i, t_q)$	The Inclusion Performance Impact (IPI) of including t_q into $\mathbf{a}_i$
$\gamma_i^{\oplus} = [w_1^{\oplus}, \ldots, w_m^{\oplus}]$	A list to store the IPIs of each task on each vehicle
$\beta_i = [\beta_1, \ldots, \beta_m]$	A vehicle ID list corresponding to the RPI list that keeps track of which task is assigned to which vehicle.
$\psi_i = [t_1, \ldots, t_\zeta]$	Candidate tasks for inclusion into v_i's task list
$\mathbf{a}_i^{\ominus k}$	Temporary task list with t_k removed
$\mho_{i,k}$	List of tasks that can replace t_k in $\mathbf{a}_i$ while respecting time constraints
SD	Swap Distance: maximum number of permissible reassignments to create a time slot for an unassigned task
r	Reduction rate of RPI-MaxAss for each additional reassignment
ϖ_i	A vector that stores the number of times each task has been removed from a vehicle v_i's task list

Table 2.5 Symbol Definitions for BW-CBBA extensions

Symbol	Definition
$\mathbf{V} = [v_1, \ldots, v_n]$	Set of n agents
$\mathbf{T} = [t_1, \ldots, t_m]$	Set of m tasks
$\mathbf{b}_i$	set of tasks assigned to v_i ordered chronologically based on when the tasks were added.
$\mathbf{p}_i$	Ordered task allocation of the $i-th$ agent v_i
ξ_k	Latest start time for task t_k
ς_{ik}	The predicted time of execution of $t_k \in \mathbf{p}_i$
f_i	The latest time at which v_i can start a task before running out of fuel
$\lvert\mathbf{p}_i\rvert$	The number of tasks assigned to v_i
$c_{i,k}$	The score that v_i computes for candidate task t_k
$\bar{c}_{ik}$	$c_{i,k}$ warped to satisfy Diminishing Marginal Gain

Algorithm 3 CBBA: Bundle Building with Non-DMG Scores [45]

procedure BUILD BUNDLE
  while $|\mathbf{p}_i| < L_t$ **do**
    for $t_q \in T \setminus \mathbf{p}_i$ **do** // Candidate tasks not already in agent i's path
      $c_{iq} = \max_l F_{iq}(\mathbf{p}_i \oplus_l t_q), \quad \forall l \leq |\mathbf{p}_i| + 1$ // Compute score for each candidate
      $\bar{c}_{iq} = G_{iq}(c_{iq}, \mathbf{b}_i)$ // Warp score to satisfy the DMG condition
      $h_{iq} = \Pi(\bar{c}_{iq} > y_{iq})$ // Check if DMG score is higher than highest bid
    end for
    $q^\star = \text{argmax}_q c_{iq} \cdot h_{iq}$ // Find candidate task with highest score
    if $\bar{c}_{iq^\star} > 0$ **then**
      $z_{iq^\star} = i$
      $y_{iq^\star} = \bar{c}_{iq^\star}$
      $\mathbf{b}_i \oplus_{end} t_{q^\star}$
      $\mathbf{p}_i \oplus_l t_{q^\star}$ where l yielded $c_{iq^\star}$
    else
      break
    end if
  end while
end procedure

Chapter 3

Distributed Task Rescheduling

This chapter introduces the first original contribution in this thesis: the extension of the PI algorithm to a novel algorithm, named PI-MaxAss. The work described in this section was published in [94] and [97] during the work for this Ph.D. thesis.

3.1 Introduction

One challenge in using teams of robots is to co-ordinate them to perform tasks while optimising one [36, 1], or more objectives [77, 47, 93]. Considering a search and rescue scenario, in which survivors need to be assisted before specified deadlines, the two main objectives are 1) to maximise the number of rescued survivors, 2) to minimise the average waiting time before their rescue [87]. The novel contribution outlined in this section, called PI-MaxAss, was devised to solve the problem of increasing the number of allocated tasks in a distributed team of agents where deadlines prevented all tasks from being assigned. Specifically, if a certain task could be reached in time by one agent only, the approach ensures that the agent selects this task. A first hypothesis: given pre-defined conflict resolution rules, if the cost of an assignment is high when the assignment could be replaced by an unassigned task, this would facilitate an increase in the number of assigned tasks when possible. A second hypothesis was that by starting from an existing task allocation, such an approach would either increase the number of allocated tasks or leave the number unchanged.

PI-MaxAss assigns costs to task assignments such as to shift task assignments among vehicles to create feasible time slots for unassigned tasks. The maximum number of reassignments can be adjusted to match performance requirements. With this method, existing task assignment solutions are iteratively improved without the need to repeat the whole task allocation procedure. The procedure follows a two-phase task assignment strategy that starts from a solution generated by an existing distributed task allocation algorithm, Performance Impact (PI) [102], that minimises average waiting time. The proposed method PI-MaxAss is used in the second stage for maximising task allocations. A simulated rescue scenario with task deadlines and fuel limits is used to demonstrate the performance of the proposed method compared with CBBA and baseline PI. Starting from existing (PI-generated) solutions, results show an up to 20% increase in task allocations using the proposed method. PI-MaxAss takes the solution generated by PI, and iteratively increases the number of allocated tasks when it is possible to do so. The advantages of this design choice are:

1. PI-MaxAss has a marginally higher runtime complexity per iteration compared with PI, therefore it is advantageous to generate an initial solution with PI.

2. PI-MaxAss guarantees a solution with higher or equal number of allocated tasks to PI.

3. PI optimises average waiting time for task allocations. This optimisation is preserved for tasks that are not reassigned with PI-MaxAss. This would hold true if the solution that PI-MaxAss starts with were optimised for different objectives, such as distance covered.

3.2 PI-MaxAss

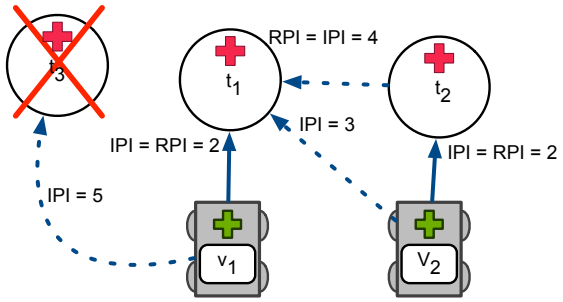
Simulated experiments have shown that the PI algorithm both allocates more tasks and optimises average waiting time better than CBBA in time critical scenarios with a low task-to-vehicle ratio [108, 102]. However, preliminary experiments showed that when there is a higher ratio of tasks to vehicles, PI can fail to allocate all tasks when it is possible to do so. Due in part to their scoring strategies, the baseline CBBA and PI do not reassign tasks when

this is necessary in order to assign additional tasks. In the search and rescue scenario the safety and rescue of survivors is a high priority; a poorer quality of solution results in fewer survivors being rescued than is possible with the available resources.

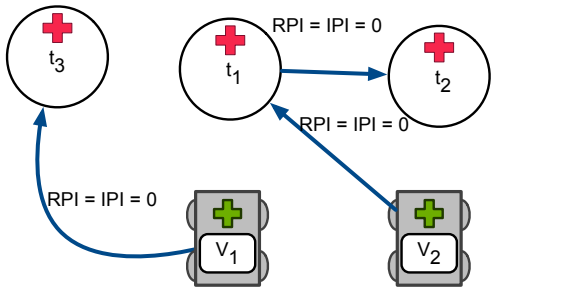
Starting from a sub-optimal assignment in which additional tasks cannot be directly included without violating time constraints, the extension PI-MaxAss presented in this thesis is able to reassign tasks to increase the total number of allocated tasks simply through a change in the computation of IPIs and RPIs. The idea introduced in this work is to attribute a high cost (RPI) to an assigned task when the release of this task can permit an additional task to be inserted within the free time created. An assignment is considered optimal and without cost if the release of any task does not permit another task to be assigned within the free time created. Likewise, a task's IPI is set to be without cost if it can be included into a task list and satisfy time constraints. During the conflict resolution phase, conflicts resolve in favour of vehicles offering the lowest RPI. Vehicles that can create a time slot for candidate tasks through the release of an assigned task therefore release that task during a conflict. The result is that tasks are reassigned and feasible time slots are created for unassigned tasks.

3.2.1 Limitation of previous methods and proposed solution

To illustrate the limitation of previous methods and the proposed solution, consider a simple scenario shown in Fig. 3.1(a) and the associated schedule on a timeline in Fig. 3.2(a). With the PI algorithm, the vehicles include tasks into their lists starting with the lowest IPI. With PI-MinAvg, v_1 first includes t_1 and v_2 first includes t_2 into their task lists. Once included, t_1 cannot be released from v_1 unless v_2 includes t_1 with a lower RPI. Likewise, t_2 cannot be released from v_2 unless v_1 includes t_2 with a lower RPI. For t_3 to be serviced before its deadline, v_1 must go to t_3 directly. However, v_1 is incapable of servicing both t_1 and t_3 and meet both of their time constraints. Task t_1 does not get reassigned to v_2 because the RPI of t_1 is lower in v_1's task list than in v_2's task list. Therefore t_3 does not get assigned. The suboptimal task allocation is due to the minimisation of waiting time performed by PI-MinAvg. The novelty in PI-MaxAss is that the cost of t_1 in v_1's task list is higher than in v_2's task list, causing t_1 to be reassigned to v_2. This creates a time slot in v_1's schedule for t_3.



(a) PI-MinAvg



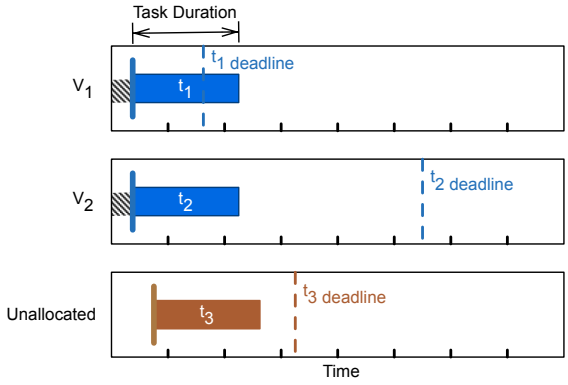
(b) PI-MaxAss

Fig. 3.1 In this scenario PI-MinAvg is unable to assign all tasks. PI-MaxAss assigns all tasks. Dotted lines connecting vehicles to tasks indicate examples of IPIs computed during the task inclusion phase. Solid lines indicate tasks assigned after reaching consensus. (a) Each task assignment is labeled with its PI-MinAvg IPI or RPI. With PI-MinAvg t_1 is assigned to v_1, t_2 is assigned to v_2, and t_3 is left unassigned. v_2 may also include t_1 if v_2 has not yet received v_1's RPI list. In this case v_2 releases t_1 during the conflict resolution phase due to a higher RPI than v_1. (b) PI-MaxAss reassigns tasks starting from the PI-MinAvg solution and creates a time slot for t_3. Each task assignment is labeled with its IPI and RPI for maximising the number of task assignments.

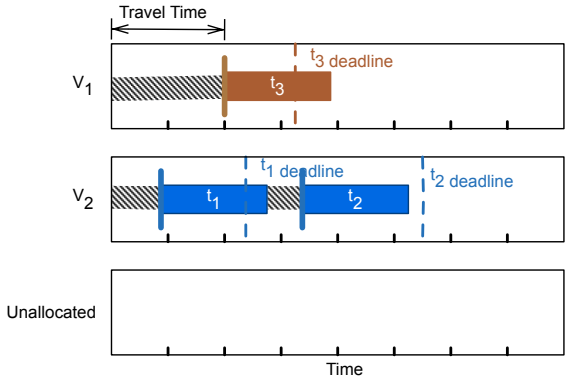
Therefore, PI-MaxAss achieves the optimal allocation illustrated in Fig. 3.1(b) and Fig. 3.2(b). Although the waiting time for t_1 and t_2 has increased in Fig. 3.2(b), this reassignment has enabled an additional task to be assigned.

3.2.2 Formal Description

With PI-MaxAss, unallocated tasks are set initially to have a fixed highest RPI-MaxAss, a constant defined as U, such that if t_q is unassigned then $w_q^{\ominus} = U$. The RPIs of assigned tasks t_k are initially set to 0, such that $w_k^{\ominus} = 0$.



(a) PI-MinAvg



(b) PI-MaxAss

Fig. 3.2 Task schedules for v_1 and v_2. A travel time is assumed between the vehicles' initial locations and between different task locations, based on the distance and speed that they can travel. A fixed task duration is also assumed. A task must be started before the deadline in order to rescue that survivor, but may end after the deadline. v_1 is the only vehicle close enough to reach t_3 in time. (a) t_1 and t_2 are optimised to minimise waiting time but t_3 is unallocated. v_1 cannot feasibly include t_3 into its schedule given t_1. (b) If t_1 is reassigned from v_1 to v_2, this creates the time slot for v_1 to include the unallocated task t_3.

The steps of PI-MaxAss follow the two phases depicted in Algorithm 1. During the task inclusion phase shown in Algorithm 2, as with PI-MinAvg, the PI-MaxAss candidate tasks for inclusion into $\mathbf{a}_i$ are those compatible with v_i's capabilities and not already in $\mathbf{a}_i$, and with an RPI-MaxAss greater than 0. The candidate tasks for inclusion into $\mathbf{a}_i$ are formally defined as

$$\psi_i = [t_1, \ldots, t_\zeta], \quad t_q \notin \mathbf{a}_i, 0 < w_q^{\ominus} \quad . \tag{3.1}$$

The IPI-MaxAss of t_q in $\mathbf{a}_i$ is formally defined as

$$w_q^{\oplus\star}(\mathbf{a}_i, t_q) = 0, \exists l \quad \forall t_z \in \{\mathbf{a}_i \oplus_l t_q\} \\ : c_{i,z}(\mathbf{a}_i \oplus_l t_q) \leq min(s_z, f_i), t_q \in \Psi_i \quad . \tag{3.2}$$

In other words, the IPI-MaxAss of the candidate task t_q is set to 0 if there exists a position l in $\mathbf{a}_i$ where the task t_q is inserted and all time constraints are met. On line 10 in Algorithm 2, IPI-MaxAss is recorded in place of IPI-MinAvg such that $w_q^{\oplus\star} = 0$ if the condition on line 6 returns true for at least one position l. The optimal position l is computed as it is for IPI-MinAvg, according to Equations (2.4) and (2.5).

Lines 13–21 in Algorithm 2 remain the same for PI-MaxAss. As the RPI-MaxAss of assigned tasks were initialised to 0, only unassigned tasks are candidates for inclusion in the first round of the task inclusion phase. RPI-MaxAss is computed on line 22 in the place of RPI-MinAvg. The steps for computing RPI-MaxAss are shown in Algorithm 4.

Candidate tasks in the computation of RPI-MaxAss follow the same contraints as the candidates in Equation (3.1) with the added constraint that the candidate task's RPI-MaxAss is greater than δ. This constraint is used to limit the number of reassignments permissible

Algorithm 4 Computing RPI-MaxAss for tasks in v_i's task list

```
1: Set RPI of tasks in a_i to 0: γ_{i,k} ← 0,    t_k ∈ a_i
2: Identify Candidate Tasks: ψ̄_i
3: for each task k in a_i do    // For each task k in task list
4:     a_i^{⊖k} = a_i ⊖ t_k    // Remove k from task list
5:     Update times c_{i,z}(a_i^{⊖k}) for tasks after t_k
6:     for each task q in ψ̄_i do    // For each candidate task q
7:         if γ_{i,q} − r > γ_{i,k} then
8:             for each position l in a_i^{⊖k} do
9:                 if a_i^{⊖k} ⊕_l t_q is feasible then    // If task k can be replaced by task q
10:                    γ_{i,k} = γ_{i,q} − r    // Compute new RPI for k
11:                    break
12:                end if
13:            end for
14:        end if
15:    end for
16: end for
```

to allocate an additional task (see 3.2.3). Candidate tasks used in the computation of RPI-MaxAss for a task t_k are formally defined as

$$\bar{\psi}_i = [t_1, \ldots, t_\zeta], \quad t_q \notin \mathbf{a}_i, \quad 0 < \delta < \gamma_{i,q} \quad . \tag{3.3}$$

The identification of candidate tasks occurs on line 2 in Algorithm 4. To compute the RPI-MaxAss of a task t_k in $\mathbf{a}_i$, first, a temporary task list $\mathbf{a}_i^{\ominus k}$ is created that is equivalent to $\mathbf{a}_i$ with t_k removed and is formally defined as

$$\mathbf{a}_i^{\ominus k} = \mathbf{a}_i \ominus t_k, \quad t_k \in \mathbf{a}_i \quad . \tag{3.4}$$

The creation of $\mathbf{a}_i^{\ominus k}$ occurs on line 4 in Algorithm 4. Next, a candidate task t_q is inserted into each position l in $\mathbf{a}_i^{\ominus k}$ to determine if there exists a position l in $\mathbf{a}_i^{\ominus k}$ in which t_q is inserted and all time constraints are met. If such a position l exists then t_k can feasibly be replaced by t_q in $\mathbf{a}_i$ and the RPI-MaxAss of t_k is computed as the RPI-MaxAss of t_q reduced by r. This computation is repeated for each task t_q in $\bar{\psi}_i$. The list of tasks $\mho_{i,k}$ that can replace t_k in $\mathbf{a}_i$ while respecting time constraints is formally defined as

$$\begin{aligned} \mho_{i,k} = \{ t_q \in \bar{\psi}_i \mid \quad & \exists l \quad \forall t_z \in \{\mathbf{a}_i^{\ominus k} \oplus_l t_q\} \\ & : (c_{i,z}(\mathbf{a}_i^{\ominus k} \oplus_l t_q) \leq min(s_z, f_i)) \} \quad . \end{aligned} \tag{3.5}$$

If a task t_k in $\mathbf{a}_i$ can be replaced by two or more candidate tasks t_q with different RPI-MaxAss, the highest RPI-MaxAss is recorded. The RPI-MaxAss of a task is formally defined as

$$w_k^{\ominus\star}(\mathbf{a}_i, t_k) = \max_{q=1}^{|\mho_{i,k}|} \{ w_q^{\ominus\star} - r \}, \quad t_q \in \mho_{i,k}, r \in \mathbb{R}_+ \quad . \tag{3.6}$$

The condition on line 7 in Algorithm 4 ensures that the feasibility of inserting t_q into $\mathbf{a}_i^{\ominus k}$ is not computed if the the resulting RPI-MaxAss of t_k is not higher than its current value. This condition reduces unnecessary computation and satisfies finding the maximum RPI-MaxAss according to (3.6). The condition on line 9 checks the feasibility of inserting t_q in position l

in $\mathbf{a}_i^{\ominus k}$ so that the computation of RPI-MaxAss on line 10 is performed only with candidate tasks that satisfy (3.5).

Fig. 3.3 illustrates how the computation of a decreasing RPI-MaxAss allows for multiple reassignments to create a time slot for an unassigned task, and signposts the path with the fewest reassignments. Fewer reassignments minimises the time to reach consensus and better maintains the original solution's optimisation for minimising average waiting time.

3.2.3 Swap Distance

In a time critical scenario such as search and rescue, it may be necessary to limit the time it takes for the distributed system to converge to a task allocation. The time to converge partly depends on the number of iterations of the algorithm until consensus. Depending on the network topology, propagating new assignments across the network may require multiple iterations affecting the total time to consensus. Therefore, with PI-MaxAss, limiting the number of reassignments permissible to assign an unassigned task is required. A maximum number of reassignments, expressed as "Swap Distance" *SD* is defined. SD is a new parameter, not present in CBBA or PI-MinAvg, introduced in PI-MaxAss to limit the maximum number of reassignments. It was empirically derived that, as a rule of thumb, a maximum of 1 or 2 reassignments provides the best trade-off between an increase in the number of allocated tasks and the increase in the number of iterations resulting from this method. Further guidance on setting SD is discussed in Section 3.4. As defined by (3.3), a candidate task in $\bar{\psi}_i$ must have an RPI-MaxAss greater than δ which limits the number of reassignments to *SD*; δ is defined as

$$\delta = U - (r * SD), \quad r < \frac{U}{SD}, SD \in \mathbb{R}_+, U \in \mathbb{R}_+ \quad . \tag{3.7}$$

In Fig. 3.3, $U = 100$ and $r = 10$. If $SD = 0$ then $\delta = 100$ resulting in no candidates for the computation of RPI, according to (3.3). As a consequence only unassigned tasks have an RPI greater than 0 and can therefore be included in the task inclusion phase according to (3.1). If $SD = 1$ then $\delta = 90$ and one reassignment is permissible for the inclusion of an unassigned

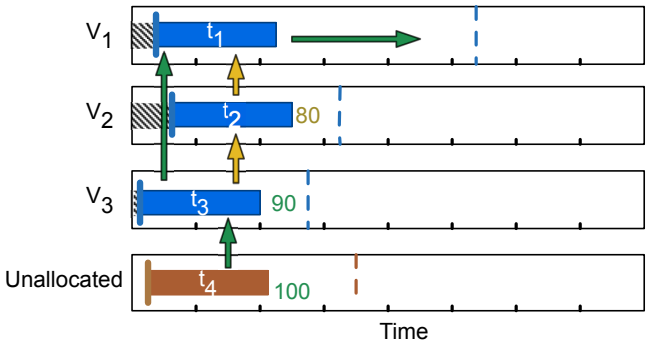


Fig. 3.3 RPI-MaxAss minimises number of changes to existing task assignments to create a time slot for an unallocated task. In this scenario it is assumed that v_3 is the only vehicle near enough to t_4 to service it in time. t_4 is unallocated and takes RPI-MaxAss $= U$ = 100. r is set as 10. t_3 can be replaced by t_4 according to (3.5) therefore t_3's RPI-MaxAss is 100 - 10 = 90 according to (3.6). t_2 can be replaced by t_3 therefore t_2's RPI-MaxAss is 90 - 10 = 80. During the task inclusion phase, v_1 can include t_3 or t_2 (without removing t_1) therefore t_3 and t_2's IPI-MaxAss are 0 according to (3.2). Given Equation (2.6), v_1 selects t_3 for inclusion as t_3 yields the greatest difference between RPI and IPI. During the communication and conflict resolution phase, v_3 releases t_3 due to having a higher RPI-MaxAss for t_3 than v_1. During the task inclusion phase, v_3 includes t_4. The decreasing RPI-MaxAss ensures that the minimal number of reassignments is selected when different options are available for the inclusion of an unassigned task.

task. In Fig. 3.3, the path that requires two reassignments in which the RPI-MaxAss of t_2 is 80 is not permissible when $SD = 1$. When $SD = 1$, t_3 does not satisfy the constraints to be in $\bar{\psi}_2$ because its RPI-MaxAss is not greater than δ, therefore the RPI-MaxAss of t_2 remains as 0. The path with two reassignments is only possible with $SD = 2$ (or higher). SD therefore restricts the tasks eligible to be candidates so that the number of reassignments is less than or equal to SD.

3.2.4 Convergence

Preliminary experiments running PI showed that two or more vehicles occasionally get caught in an infinite cycle exchanging the same tasks. In order to avoid infinite cycles and to guarantee convergence, the solution proposed here is to limit the number of times that a vehicle can remove the same task from its list before it no longer attempts to include it. This proposed approach will iteratively remove tasks involved in an infinite cycle from an

agent's search space. With the assumption that the number of tasks is finite, in the worst case, all tasks will be removed from the agent's search space, at which point convergence is guaranteed because with no assignments there can be no conflicts. A maximum limit on removals Υ where $\Upsilon \in \mathbb{Z}_+$ can be set. This precaution may prevent those tasks that are being repeatedly exchanged from being allocated optimally, however it ensures that the system can converge. A vector ϖ_i is used to store the number of times each task has been removed from a vehicle v_i's task list. During the conflict resolution Phase when a task t_k has been removed from v_i's task list: $\varpi_{i,k} = \varpi_{i,k} + 1$. During the task inclusion phase, a task t_k is considered a candidate in ψ_i for inclusion if $\varpi_{i,k} < \Upsilon$ is satisfied.

3.2.5 Complexity

To assess the computational complexity of running PI-MaxAss on one vehicle, the method used in [108] is followed. In [108], the computational complexity of PI-MinAvg is determined to be polynomial. The complexity is dominated by the computation of IPI-MinAvg during the task inclusion phase and it is defined in [108] as

$$O((m_i - |\mathbf{a}_i|)|\mathbf{a}_i|^2)\vartheta_y\sigma \quad , \tag{3.8}$$

where $|\mathbf{a}_i|$ represents the cardinality of the task list $\mathbf{a}_i$. m_i is the capacity of vehicle v_i. A maximum number $m_i - |\mathbf{a}_i|$ tasks can be added into a vehicle's task list during each iteration of the algorithm. σ denotes the complexity of computing the time cost of a task. ϑ_y denotes the number of tasks that are not yet in the task list and meet the compatibility constraints. ϑ_y is equivalent to the cardinality of candidate tasks $|\psi_i|$ as defined in this thesis. In the experiments conducted in this study, no hard limit was imposed on the number of candidate tasks. However, such a parameter could be introduced to limit the computational cost of the task inclusion phase.

The complexity of PI-MaxAss is dominated by the computation of each task's RPI-MaxAss in vehicle v_i's task list, as shown in Algorithm 4. The first step in the outer loop (for each task in vehicle v_i's task list) is to remove a task and adjust the times of the remaining

tasks in the temporary task list $\mathbf{a}_i^{\ominus k}$; the complexity is $|\mathbf{a}_i^{\ominus k}|(|\mathbf{a}_i^{\ominus k}|+1)\sigma/2$. Within the inner loop, the task times of each task starting from the position of the included task are computed: $|\mathbf{a}_i||\bar{\psi}_i|(|\mathbf{a}_i^{\ominus k}|+1)((|\mathbf{a}_i^{\ominus k}|+1)+1)\sigma/2$. Altogether that is $|\mathbf{a}_i^{\ominus k}|(|\mathbf{a}_i^{\ominus k}|+1)\sigma/2+|\mathbf{a}_i||\bar{\psi}_i||\mathbf{a}_i|(|\mathbf{a}_i|+1)\sigma/2$. This simplifies to

$$O(|\mathbf{a}_i|^3|\bar{\psi}_i|\sigma/2) \quad . \tag{3.9}$$

The RPI-MaxAss computation has a higher complexity than the RPI-MinAvg computation, but is equivalent to the complexity of computing IPI-MinAvg.

3.3 Experiments

This section presents the results of numerical simulations conducted to test the performance of the proposed PI-MaxAss compared with the performance of PI-MinAvg and CBBA when maximising allocated tasks in scenarios with time constraints. CBBA is an established benchmark for comparison in distributed task allocation problems and therefore provides a useful metric for general comparisons with similar algorithms. Thus, the evaluation of the proposed method is performed by comparison with CBBA using a range of parameter settings. All simulations were performed in MATLAB on a computer with 2.5 GHz Intel Core i5 CPU and 8 GB of RAM.

3.3.1 Scenario and Simulation Setup

To test the robustness of the proposed approach, the same types of scenarios as in [108, 102] were used. These include scenarios with a variety of different parameters including task and vehicle numbers, and network topologies. Moreover, the parameter settings are extended in this study to include a more challenging high task-to-vehicle ratio, and to include fuel constraints on vehicles. Preliminary experiments revealed that changing other parameter settings such as the starting positions of the vehicles e.g. all vehicles starting from the same position, did not significantly affect the number of task allocations. The setup uses a rescue

team equally split into two vehicle types with different functions. One vehicle type provides medicine, the other provides food. All tasks are considered to have equal priority to facilitate a clearer analysis of the task allocation maximisation process. However, a range of priorities could be introduced in future extensions of the algorithm through an ordering of candidate tasks.

The scenario specification, summarised in Table 5.1, is as follows: the vehicles' speeds are assumed to be constant and are set to 30m/s and 50m/s respectively. The assumption is that the speeds are used for the purpose of determining travel time. Further details such as acceleration are not modelled. The survivors are likewise equally split into those requiring food and those requiring medicine. The medicine tasks last for a duration of 300 seconds, the food tasks last 350 seconds. The deadlines for starting each rescue are uniformly distributed on a timeline between 0 and 2000 seconds. The mission takes place in a 3D space spanning 10 000m x 10 000m x 1000m. The tasks are randomly placed in a 3D space, and vehicles on the 2D ground space, with coordinates drawn from uniform distributions. The battery limit of each vehicle is set randomly between 1000 and 2000 seconds. Random initialisations are pseudorandom and are generated with a random number generator in MATLAB. Given the random initialisation of task and vehicle locations and deadlines, it is sometimes impossible for some tasks to be started by any vehicle before its deadline. In these simulations, all task information is available to all vehicles up front. The task allocation procedure is performed

Table 3.1 Scenario Specification

	Medicine	Food
Vehicle Speed	30m/s	50m/s
Vehicle Battery	Between 1000 and 2000 seconds	
Vehicle Start Position	10 000m x 10 000m x 0m ground space	
Task Duration	300 seconds	350 seconds
Task Deadline	Between 0 and 2000 seconds	
Task Location	10 000m x 10 000m x 1000m 3D space	

before any tasks are executed. Previous studies have demonstrated that the PI algorithm is effective at allocating new tasks online [103]. Results are computed as averages over 50 runs, where a run is a completed experimental simulation of the task allocation procedure. For each run, the time to convergence is determined by the number of iterations of the algorithm until the agents reach consensus.

3.3.2 Simulation Results

PI-MinAvg vs. PI-MaxAss

Fig. 3.4 compares the PI-MinAvg solutions with the PI-MaxAss solutions that are initialised with the PI-MinAvg solution. A row formation was used for these experiments and a Swap Distance of 2 ($SD = 2$) was set. Fig. 3.4(a) shows the percentage of runs where PI-MaxAss increased the number of allocated tasks from the PI-MinAvg solution. Fig. 3.4(b) shows the corresponding average percentage change and standard deviation of number of allocated tasks when PI-MaxAss changed the number of allocated tasks.

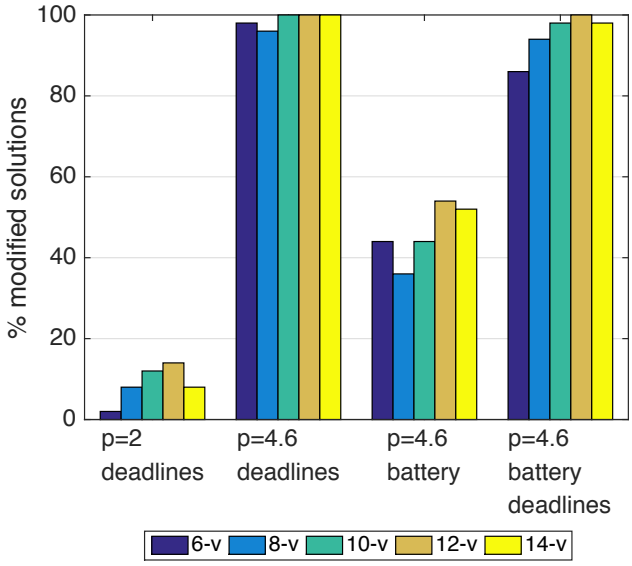
Fig. 3.4 shows both the results using the same experimental setup as in [108, 102] with a task-to-vehicle ratio of 2 to 1 (ratio $p = 2$), deadlines for each task and without battery limit time constraints, and results using a task-to-vehicle ratio $p = 4.6$ with task deadlines only, vehicle battery limits only, and combined task deadlines and battery limits, respectively. Ratio $p = 4.6$ was selected to test the system approaching maximum capacity. In [108, 102] experimental results showed that PI-MinAvg was capable of finding a solution that maximised the number of allocated tasks in most cases. The ratio $p = 2$ results in 3.4(a) reflect these findings. For each of the 5 setups with ratio $p = 2$, PI-MaxAss increased the number of allocated tasks from the PI-MinAvg solution; in the best case 14% of the runs were improved upon. In each run that PI-MaxAss increased the number of allocations, starting from PI-MinAvg with $p = 2$, one extra task was allocated. The results for ratio $p = 4.6$ show that when the system is approaching maximum capacity, i.e. when the order and allocation of tasks is critical to optimise number of allocated tasks, PI-MaxAss increased the number of task allocations in approximately half the runs with battery only time constraints and in up to

100% of runs with task deadlines. Up to 3 extra tasks were assigned in runs with battery only time constraints. Up to 8 extra tasks were assigned in runs with task deadlines with ratio $p = 4.6$. In one such instance, PI-MaxAss increased the number of allocated tasks from 44 to 52 out of 56 tasks, where 4 tasks were impossible to allocate from the outset due to their relative positions and deadlines. In other words, PI-MaxAss facilitated an 18% increase in allocated tasks achieving the maximum allocation. In another instance, a 20% increase was achieved by increasing the number of allocations from 35 to 42 out of 46 tasks.

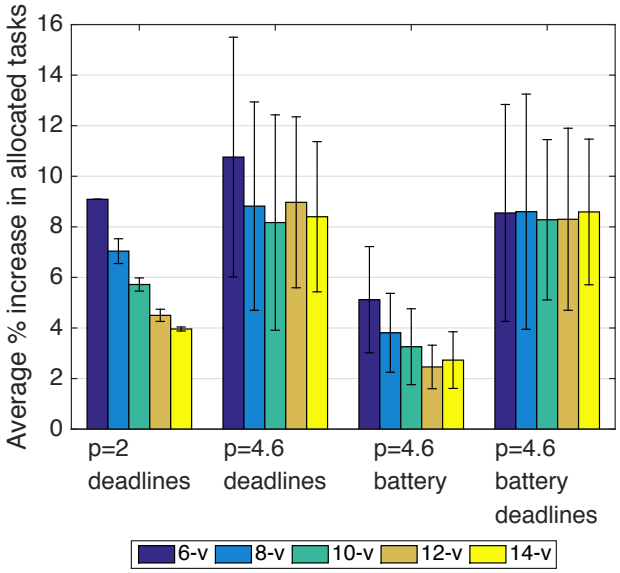
Out of the total number of experimental runs covering different ratios and constraints, only in 6 cases PI-MaxAss modified the solution by reassigning tasks without increasing the total number of assigned tasks. In all other instances that the solution was modified, the number of allocations was increased.

Swap distance parameter comparison

Fig. 3.5 shows the results of a comparison between the performance of CBBA, PI-MinAvg, and PI-MaxAss with Swap Distance set between 1 and 4. The performance with regards to number of allocated tasks and number of iterations until convergence is presented. The total number of iterations for one simulation is determined by the last time an allocation change was made, either through inclusion or removal. As the PI-MaxAss solutions are initialised with the solutions from PI-MinAvg, the number of iterations for a run of PI-MaxAss is the sum of iterations taken for PI-MinAvg and PI-MaxAss, so PI-MaxAss will necessarily be at least as high as PI-MinAvg in all instances. Fig. 3.5(a) is a notched box and whisker plot with outliers [67] that shows the total number of allocated tasks for each algorithm. Each box represents the interquartile range that encompasses the center 50% of the data. The line on the notches represents the median of the data. Fig. 3.5(b) is the same type of plot that shows the corresponding total number of iterations for each algorithm. The statistical significance between different Swap Distances for numbers of allocations and iterations was evaluated using a Wilcoxon Rank Sum Test, equivalent to the Mann-Whitney U-test. With a null hypothesis that the two samples are independent, a failure to reject the null hypothesis occurs at 5% significance. The tests indicated that increases in allocated tasks between PI-MinAvg,



(a)



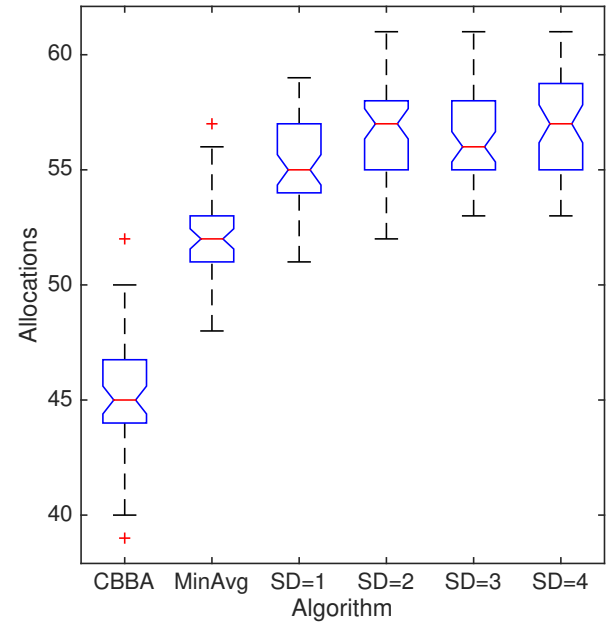
(b)

Fig. 3.4 For each scenario, the simulations were tested for an increasing number of vehicles and tasks. Vehicle numbers were 6, 8, 10, 12 and 14. For the ratio $p = 2$ the number of tasks were 12, 16, 20, 24 and 28, and with ratio $p = 4.6$ task numbers were 28, 36, 46, 56, 64. Ratio $p = 2$ was tested with task deadlines, ratio $p = 4.6$ was tested with task deadlines, with battery limits only, and with battery limits and task deadlines, respectively. In (a) each bar shows the percentage of solutions over 50 experimental runs that PI-MaxAss assigned additional tasks starting from PI-MinAvg solution. (b) shows the corresponding average percentage change with whiskers representing the standard deviation in number of allocated tasks when PI-MaxAss changed the number of allocated tasks.

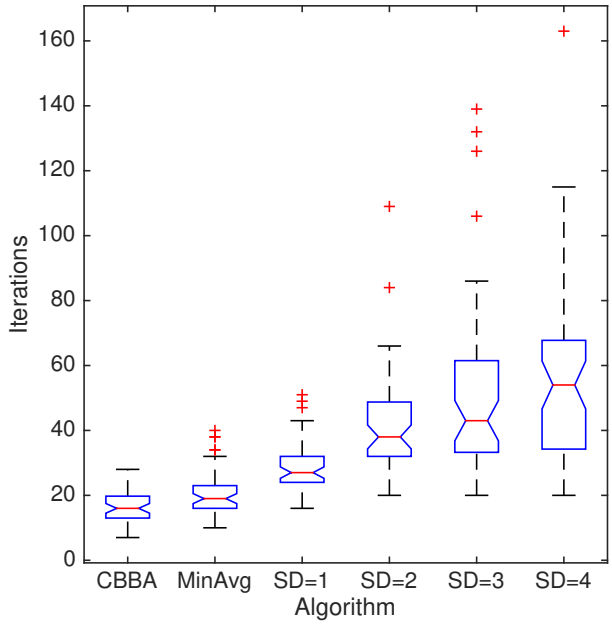
PI-MaxAss with $SD = 1$ and $SD = 2$ are statistically significant. Meanwhile, the increase in the number of allocated tasks between PI-MaxAss with $SD = 2$, $SD = 3$, and $SD = 4$ were not significant. The increase in the number of iterations between PI-MinAvg, PI-MaxAss with $SD = 1$, $SD = 2$, and $SD = 3$ were shown to be significant, while the difference between $SD = 3$ and $SD = 4$ was shown to be not significant. Therefore, for $SD = 3$ and $SD = 4$ there is an increase in iterations without a significant increase in task allocations compared with $SD = 2$. Table 3.2 shows that when the Swap Distance is limited to 1, an average of 3 extra tasks are allocated from the PI-MinAvg solution (Shown in Table 3.3) and the number of iterations has 95% confidence of being between the intervals 7.86 and 9.42 (not counting the iterations for PI-MinAvg). The trade-off is just over 1 fewer allocated tasks on average compared with $SD = 2$. As the Swap Distance increases, the confidence intervals for the number of iterations also widen.

Table 3.2 Average task allocations and iterations performance of PI-MaxAss over 50 simulations, with standard deviation and confidence intervals for iterations

Deadlines				
14-v 64-t	$SD = 1$	$SD = 2$	$SD = 3$	$SD = 4$
Allocations	57.7	58.8	59	59.2
Iterations	8.9	25.7	40.3	47.2
Std dev	3.0	23.8	27.0	26.0
95% confidence	8.1-9.8	19.0-32.5	32.6-48.0	39.8-54.6
Batteries and Deadlines				
14-v 64-t	$SD = 1$	$SD = 2$	$SD = 3$	$SD = 4$
Allocations	55.1	56.4	56.7	56.8
Iterations	8.6	20.4	30.3	34.5
Std dev	2.7	15.2	25.8	26.1
95% confidence	7.9-9.4	16.0-24.7	23.0-37.7	27.1-41.9



(a)



(b)

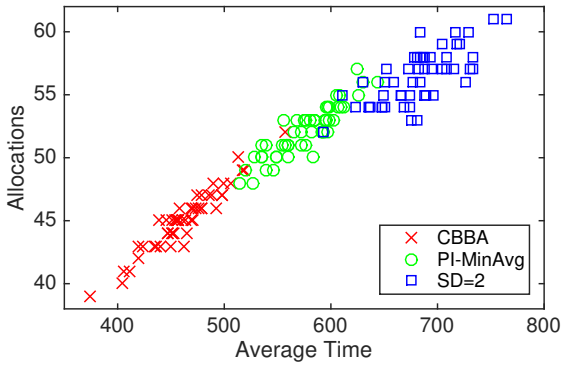
Fig. 3.5 Comparison of CBBA, PI-MinAvg, PI-MaxAss with Swap Distance 1, 2, 3 and 4 on number of allocated tasks and iterations for the 14-vehicle 64-tasks scenario with battery limits and task deadlines. The plus symbols represent outliers. a) A notched box plot of the total number of allocated tasks for each of the 50 runs for each algorithm. b) A notched box plot of the number of total iterations for each of the 50 runs for each algorithm.

Table 3.3 Task allocations and iterations performance of PI-MaxAss with Swap Distance = 2 starting with PI-MinAvg solution compared with PI-MinAvg and CBBA over 50 simulations (Average and Standard Deviation)

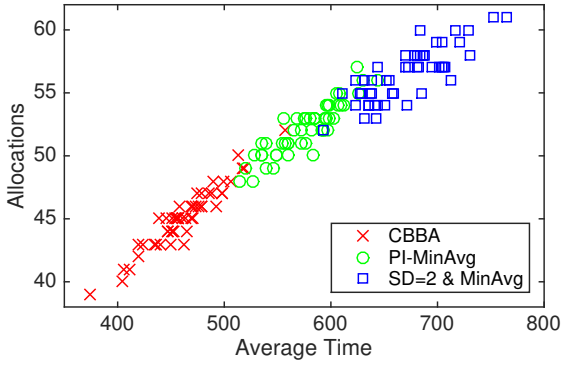
			Task Allocations (Avg)						Iterations (Avg)					
Ratio	Vehicles	Tasks	CBBA	std dev	MinAvg	std dev	MaxAss	std dev	CBBA	std dev	MinAvg	std dev	MaxAss (only)	std dev
	6	12	10.4	0.9	11.4	0.9	11.4	0.9	3.8	1.4	7.2	2.2	0.1	0.7
$p = 2$	8	16	13.8	1.2	15.2	1.0	15.3	1.0	5.6	1.9	11.3	2.8	0.3	1.1
deadlines	10	20	17.4	1.4	19.2	1.0	19.3	0.8	7.3	2.1	15.3	3.6	0.8	2.2
	12	24	20.9	1.4	22.9	0.9	23.1	0.9	8.6	2.6	21.7	6.1	1.3	3.3
	14	28	24.4	1.6	26.9	1.0	26.9	0.9	12.1	3.3	28.5	6.5	0.8	2.7
	6	28	19.3	1.7	21.9	1.5	24.2	1.6	4.8	1.5	6.8	2.1	6.3	2.3
$p = 4.6$	8	36	25.2	1.9	29.5	2.1	31.9	1.7	6.9	2.7	11.8	3.4	9.4	6.4
deadlines	10	46	32.3	2.2	38.2	2.1	41.3	2.0	10.2	2.9	16.7	5.0	19.8	19.5
	12	56	39.1	2.9	46.7	2.5	50.8	2.3	13.3	4.1	21	12.6	16.9	13.5
	14	64	45.3	2.8	54.3	2.6	58.8	2.3	15.6	4.7	26.3	7.8	25.7	23.8
	6	28	24.5	1.9	24.6	1.8	25.2	1.8	6.0	2.4	4.9	2.8	1.9	2.2
$p = 4.6$	8	36	32.9	2.1	33.4	2.1	33.8	1.9	9.6	3.9	8.2	3.8	2.0	2.6
battery	10	46	42.0	2.6	42.8	2.1	43.4	2.0	12.4	4.7	9.7	4.3	3.5	5.9
	12	56	51.2	2.4	51.9	2.2	52.6	2.1	18.6	7.6	14.6	8.5	3.2	3.9
	14	64	59.7	2.3	60.5	2.2	61.3	2.1	23.6	7.9	16.2	8.5	4.6	3.9
	6	28	19.0	2.0	21.0	1.6	22.5	1.4	4.4	1.7	5.0	1.5	7.4	12.2
$p = 4.6$	8	36	24.6	1.7	28.1	1.5	30.4	1.6	7.7	2.7	9.8	6.6	8.8	7.9
battery	10	46	32.0	2.4	36.6	2.1	39.5	2.1	9.7	2.9	11.9	4.7	10.9	6.9
deadlines	12	56	38.8	2.4	44.7	2.3	48.4	2.0	12.7	4.6	20.7	16.3	14.7	12.0
	14	64	45.0	2.3	52.1	2.0	56.4	2.2	16.5	4.9	21.1	7.6	20.4	15.2

Average time comparison

Fig. 3.6(a) plots a comparison of the average waiting time and allocations for each run. Fig. 3.6(b) plots the same results where $SD = 2$ & MinAvg shows the effect of switching back to optimising waiting time after increasing allocated tasks with PI-MaxAss. Here, PI-MinAvg was initialised with the solution of PI-MaxAss. Average waiting time logically increases as more tasks are performed. This increase is reflected in the graphs that show a proportional increase in average waiting time between CBBA, PI-MinAvg and PI-MaxAss. Fig. 3.6(b) shows that average waiting time can be optimised with PI-MinAvg after allocations have increased with PI-MaxAss. In 32 out of the 50 runs, the average waiting time is reduced,



(a)



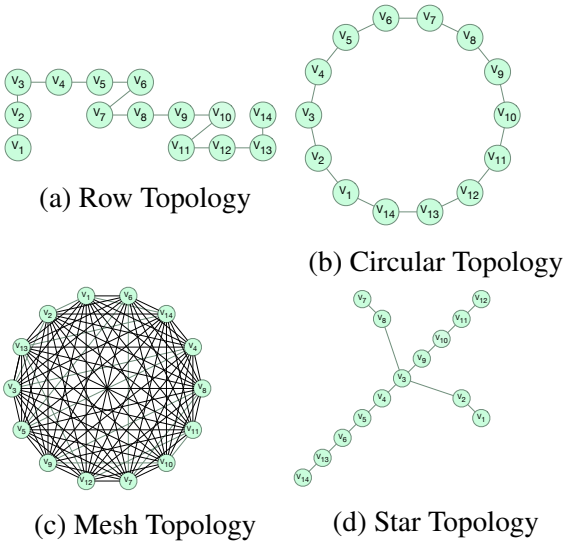
(b)

Fig. 3.6 Scatter graphs comparing the performance of CBBA, PI-MinAvg, and PI-MaxAss with Swap Distance = 2, with respect to average waiting time for 50 runs. Each plot represents the final average waiting time of all assigned tasks for one run. In b) PI-MinAvg was run starting from the solution of PI-MaxAss to show that average waiting time can be further optimised once additional tasks have been assigned. An improved average waiting time is indicated by points shifted to the left for SD=2 & MinAvg compared with SD=2.

in the best case by 63 seconds with PI-MinAvg. In this instance 4 extra tasks had been allocated with PI-MaxAss. The improvement in waiting time was achieved with 9 iterations of PI-MinAvg. The average iterations for the second round of PI-MinAvg was 6.1 over the 50 runs.

Topology comparison

Changing topologies are inherent to dynamic environments with moving vehicles. It is therefore informative to assess how the proposed method performs across different topologies [14].



(a) Row Topology

(b) Circular Topology

(c) Mesh Topology

(d) Star Topology

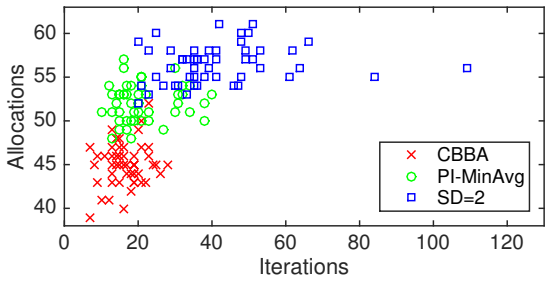
Fig. 3.7 Network topologies that the system was tested with.

Fig. 3.7 illustrates with non-directed graphs the different network topologies under which the system was tested.

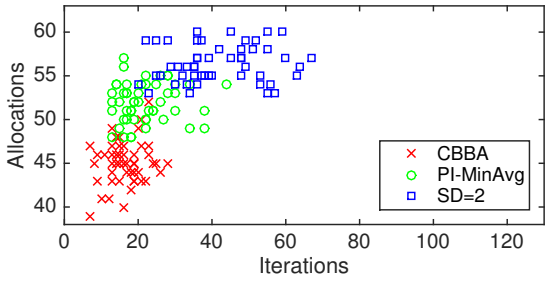
Fig. 5.1 shows the results of comparing different vehicle formation topologies on the number of allocated tasks and iterations. The row topology, circular topology, the fully connected topology and the star topology illustrated in Fig. 3.7 are compared. The number of allocated tasks is consistent across topologies for CBBA and similar across topologies for PI-MinAvg and PI-MaxAss with $SD = 2$. Notable differences are the reduced number of iterations for each algorithm with the fully connected topology and the relative increase in iterations for the star topology for each algorithm.

3.4 Discussion

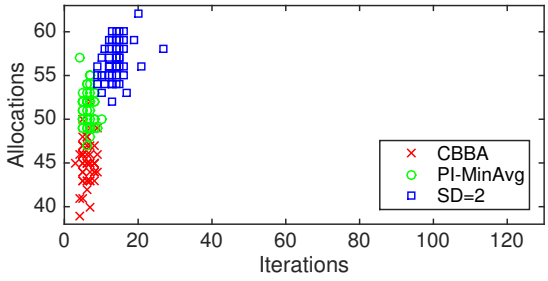
The results showed that PI-MaxAss can significantly increase the total number of allocated tasks starting from a sub-optimal solution. There is a trade-off between computation time and solution quality that should be considered depending on the application [77]. Note that computation time here is represented by the number of iterations, while in practice the processing speed and the communication speed of the agents will determine how long an iteration lasts. If extra computation time is available, the results show that switching



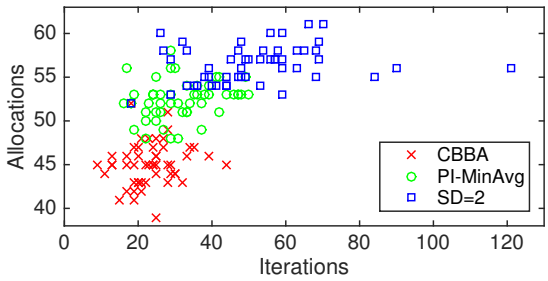
(a) Row Network



(b) Circular Network



(c) Fully Connected Network



(d) Hybrid Star Network

Fig. 3.8 Comparison of the performance over 50 runs of CBBA, PI-MinAvg, PI-MaxAss with Swap Distance 2 for 14-vehicle 64-task battery and deadlines scenario, with respect to number of allocated tasks and iterations over different network topologies, row, circular, mesh and star topologies.

optimisation objectives from minimising average waiting time to maximising task allocations can break the solution out of local optima and further optimise the task allocation without reducing the quality of the solution. After more tasks have been included, the quality of the solution can then be optimised further with few iterations by switching back to the time minimisation method. This switching strategy as described in [77] exploits the high optimisation performance of single-objective search algorithms for a bi-objective problem, while remaining flexible and modular.

In the cases where PI-MinAvg was able to reach an optimal or near optimal solution with regards to the number of allocated tasks, such as the two tasks-per-vehicle scenario, PI-MaxAss made few or no improvements on the PI-MinAvg solution and accordingly the computation time was not unnecessarily increased. These results further support the switching strategy [77] which increased computation time only when the solution could be improved by the proposed method PI-MaxAss.

The results show that a Swap Distance limited to 1 is preferable when a reliably low number of iterations is required while still providing a significantly higher number of allocated tasks. A higher Swap Distance can be used if the extra computation time is available to increase the likeliness of finding a better solution. On the other hand, although PI-MaxAss is guaranteed not to decrease allocations starting from an initial task allocation, it cannot be guaranteed that PI-MaxAss with a higher swap distance finds an equal or higher task allocation than a lower swap distance.

For the scenarios tested, the Swap Distance set to 3 and 4 did not significantly increase the allocations despite the correlated increase in iterations. For each additional task reassignment, the new task allocations are propagated through the network of vehicles, and this can take several iterations depending on the network topology meaning that, as the number of reassignments increases, so do the number of iterations. It is also likely that the number of instances where 3 or 4 reassignments are required are fewer than those requiring 1 or 2 reassignments. This may result in an insignificant increase in task allocations along with a relatively high increase in number of iterations.

PI-MaxAss was shown to be effective at increasing allocated tasks when the time constraint was on vehicle battery limits only. In these cases, the extra flexibility in the possible ordering of task allocations meant that PI-MinAvg was more likely to find an optimal solution, however PI-MaxAss increased the allocations in about half of the runs, a noteworthy proportion.

In 0.3% of 2000 runs, PI-MaxAss modified the solution by reassigning tasks without increasing the total number of assigned tasks. This may happen because an additional task allocation attempt may be inhibited if a time slot created to assign a new task is instead filled by a task later in that reassignment sequence.

Tests with different topologies provided strong evidence that the number of allocated tasks is independent of the specific topology. The number of iterations required to reach consensus, on the contrary, appears to vary according to the type of topology. The increase in iterations is due to information requiring multiple iterations or 'hops'to reach all vehicles when the network is not fully connected. In general, the longer the network diameter i.e. the shortest path between the two most distant vehicles, the longer the system takes to reach consensus.

The task shifting effect of PI-MaxAss is similar to the theoretical task swap loop methods described and analysed in [109, 62, 89, 60, 61]. Compared with these methods, PI-MaxAss has the advantage that it does not require distinguishing roles. Furthermore, PI-MaxAss does not require finding a complete swap loop to reassign tasks. As opposed to the task swap loop methods, with PI-MaxAss the last task reassignment in the sequence need not be assigned to the vehicle that started the sequence. By following the task swap loop strategy, the created time slot is more likely to be filled by the task being reassigned from another vehicle, inhibiting the assignment of an additional unassigned task. A final distinction is that the objective of PI-MaxAss is to increase the number of task assignments within vehicles' schedules, whereas the costs being minimised in [62, 60, 61] are non-specific, and the problem being addressed considers vehicles that can be assigned one task each, at most.

3.5 Conclusion

In this chapter, an effective algorithm that allows for simple and efficient reassignment of allocated tasks is proposed and analysed to improve the task allocation solution of a previous method for task allocation. The novel idea is to allow vehicles to re-allocate tasks to create a feasible space for unallocated tasks by taking advantage of existing schedule space. Simulations showed a noteworthy increase in performance, measured as the total number of allocated tasks, making the method appealing when this objective is a priority. An increment in the number of iterations appeared proportionate to the gain in performance. Experimental results confirmed that the proposed algorithm can be applied beneficially to PI, thus opening the possibility of integration to other implementations.

While PI was shown to perform well towards the optimisation objectives of minimising average time and maximising allocated tasks, the limitations of the algorithm are that it does not provide performance guarantees on convergence like CBBA. When applied to a system for which one iteration of the algorithm is expensive, a performance guarantee on convergence is a desirable feature. In the next chapter, a fast convergence approach that maintains a high number of task allocations is introduced as an extension to CBBA.

Chapter 4

Fast Convergence

4.1 Introduction

This chapter introduces a new distributed consensus procedure for consensus-based task allocation algorithms that optimises convergence time, i.e. the time required for the network of agents to agree on a task allocation. In distributed multi-agent task allocation problems, the time to find a solution and a guarantee of reaching a solution, i.e. an execution plan, is critical to ensure a fast response. In real time environments such as a rescue mission, a fast convergence time to a solution is an essential quality of an algorithm as any time spent on computing a global task allocation is time not spent rescuing. Algorithms, such as CBBA, employ the use of bids that can vary based on an agent's schedule. Changes to the agent's task list can therefore result in changes to the bids that the agent places. Additionally, when the network topology is sparsely connected, it can take multiple iterations of the algorithm before an agent receives information that it has lost a bid. It may therefore require many iterations to resolve all conflicting task allocations. This is a key limitation which becomes an increasing problem as the number of agents and the number of tasks increases.

While PI has been shown to produce better quality results under the scenarios tested with metrics such as average waiting time, the convergence guarantees of CBBA are much stronger than for PI. For this reason, CBBA is used as the baseline algorithm from this chapter onwards. The task allocation algorithm consists of two phases, in the first phase

agents build their individual schedules using a score function. In the second phase, agents resolve conflicts based on the bids they place on their selected tasks. This chapter investigates the hypothesis that in scenarios with a high task-to-agent ratio, the time to reach consensus can be reduced by removing the variability of bids. At the same time, the overall solution quality can be maintained by using appropriate heuristics in the first phase. The second key contribution in this thesis is a novel approach to stabilise the convergence process and reduce the time to reach a solution for algorithms such as CBBA. The key idea is to resolve task allocations among agents using a rank-based conflict resolution. A second advantage is that this method enables different agents to construct their task schedules using any insertion heuristic, and still guarantee convergence. Simulation results demonstrate that the proposed approach, as an extension of BW-CBBA, can allocate a greater number of tasks in a shorter time than the baseline BW-CBBA.

4.2 CBBA with Fast Convergence Design

This section introduces the rank-based conflict resolution method that reduces the time to convergence, implemented in this study as a modification to CBBA. The insertion heuristics used in combination with the rank-based conflict resolution to demonstrate the proposed method's performance are also detailed in this section. The proposed method is not limited to CBBA but may be implemented into similar distributed consensus-based task allocation algorithms.

4.2.1 Rank-based Conflict Resolution

In standard task allocation algorithms, bids on task assignments give an indication of the optimality of an assignment with respect to an optimisation objective. When conflicting assignments occur, the agent that can perform the task most optimally keeps the assignment. This process requires that bids be comparable and therefore that agents must share a function to assign scores to their assignments. The novelty in this study is to introduce bids that are invariant to factors such as the agent's path and score function. Constant bids add stability

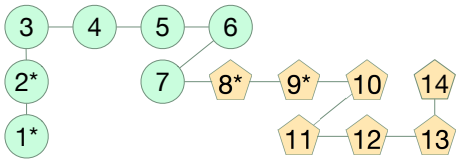
to the convergence process and therefore speed up the rate of convergence. Additionally, this method enables agents to simultaneously use different score functions from each other. As a result of losing information from bids, a trade-off is the possible reduction in quality of the task allocation with respect to the objective being optimised by the score function. In this study, the number of allocated tasks and the time to convergence are considered as the highest priority optimisation objectives, and it is therefore worth a possible reduction in optimality of secondary objectives, such as distance covered by the agents. Future work may look at autonomously adapting the task allocation method in line with the most appropriate optimisation objective given the problem domain.

Inbuilt into CBBA's conflict resolution phase is a tie-breaking heuristic based on agent identification numbers [17]. This unique numerical ID is initialised at the outset as the agent's index. When a tie occurs between bids on the same task, the agent with the lowest index wins the task. To implement conflict resolution based on agent ranking requires therefore simply that agents' bids are made to be identical at all times. The modification to the bundle building phase is shown in Algorithm 5 line 5 where a constant bid value defined as *constantBid* is applied to all bids. It is worth noting that a constant bid value satisfies the condition of DMG in equation (2.12) and therefore preserves CBBA's guarantee of convergence.

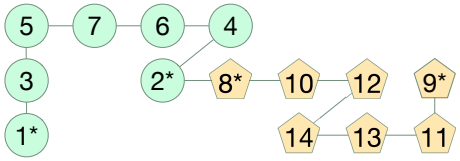
A key feature of this approach is that the distribution of rank is transitive i.e. every agent is either dominant or submissive relative to every other agent. As a consequence, agents lose conflicts only to agents of higher rank. Consider that the relative rank of each agent matches its index such that v_1 is the highest ranked and v_m the lowest ranked agent. v_1 will win all conflicts on tasks that it selects from $\mathbf{T}$. v_2 will win all conflicts on tasks that it selects from $\mathbf{T} \setminus \mathbf{b}_1$. v_i will win all conflicts on tasks that it selects from $\mathbf{T} \setminus \mathbf{b}_h, \quad \forall h \in \{1, \ldots, i-1\}$. By selecting only tasks that have not been included by higher ranking agents, an agent is ensured to have winning bids, because lower ranking agents cannot challenge that. When there are no more conflicts, the system converges. A network where agents are ranked in topological order, such as in Fig. 5.1(a), will propagate more efficiently the assignments of higher ranked agents to lower ranked agents such as to reduce the number of conflicts, compared with a network where agents are not ranked in topological order such as in Fig. 5.1(b).

Algorithm 5 CBBA: Bundle Building with EDF and agent rank bidding

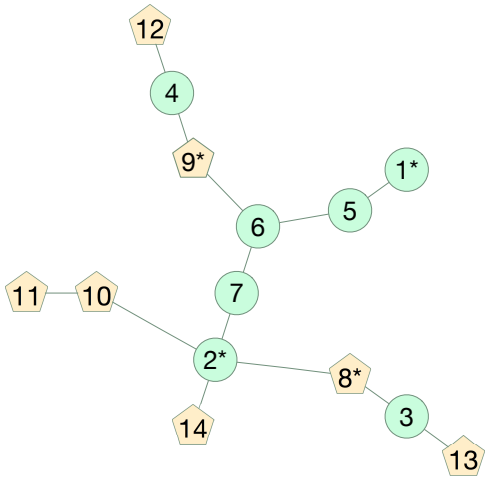
procedure BUILD BUNDLE
  while $|\mathbf{p}_i| < L_t$ **do**
    for $t_q \in T \setminus \mathbf{p}_i$ **do**
      $c_{iq} = \max_l F_i(\mathbf{p}_i \oplus_l t_q), \quad \forall l \leq |\mathbf{p}_i| + 1$
      $\bar{c}_{iq} = constantBid$ // Set score to a predefined constant
      $h_{iq} = \Pi(\bar{c}_{iq} > y_{iq})$
    end for
    $q^\star = \text{argmin}_q \xi_q \cdot h_{iq}, \quad \forall c_{iq} > 0$ // Find task q with earliest deadline
    if $\xi_{q^\star} > f_i$ **then** // If q's deadline is later than agent's fuel time
      $q^\star = \text{argmax}_q c_{iq} \cdot h_{iq}$ // Select task with highest score
    end if
    if $\bar{c}_{iq^\star} > 0$ **then**
      $z_{iq^\star} = i$
      $y_{iq^\star} = \bar{c}_{iq^\star}$
      $\mathbf{b}_i \oplus_{end} t_{q^\star}$
      $\mathbf{p}_i \oplus_l t_{q^\star}$ where l yielded $c_{iq^\star}$
    else
      break
    end if
  end while
end procedure



(a) Ordered Row Topology



(b) Unordered Row Topology



(c) Hybrid Topology

Fig. 4.1 Network topologies that determine the communication links between agents with two agent types (circles and pentagons). Agent indexes correspond to the agents' ranks. Agents 1-7 service medicine tasks and agents 8-14 service food tasks. Agents with or without a star (*) employ different insertion heuristics as explained in section 4.3.1.

The topology of the network is a determining factor in time to convergence. If agents bidding on the same tasks are not directly connected, such as in Fig. 5.1(c) where agents of the same type are connected through agents of a different type, it may take many iterations to receive bids on conflicting assignments and therefore longer to resolve conflicts and converge. A finding in section 4.3 is that the proposed consensus strategy is most effective at reducing convergence time with the ordered row topology (Fig. 5.1(a)) and least effective with the unordered hybrid topology (Fig. 5.1(c)).

4.2.2 Earliest Deadline First Task Inclusion

A main benefit of decoupling scores from bids, as shown by BW-CBBA, is the capability to match more closely the agent's internal decision making process to the optimisation objective, while maintaining convergence guarantees. This extension was shown to yield higher quality task allocations than baseline CBBA regardless that the communicated bids were required to be approximated [45].

EDF is a well known scheduling algorithm in which tasks with the earliest deadlines are given highest priority. EDF has recently theoretically and empirically been shown to be fast and effective at maximising the number of allocated tasks in a similar scenario [68]. An inclusion strategy such as EDF can cause a high number of conflicts as all agents prioritise tasks in the same order. By applying EDF task inclusion to a subset of agents, with the remaining agents using a different strategy, the number of conflicts can potentially be reduced and therefore speed up convergence compared with all agents using EDF (this is tested in section 4.3). EDF is implemented on line 8 of Algorithm 5. The best task, with index $q^\star$, is selected as the task with the earliest deadline for which h_{iq} evaluates as true.

If the agent's fuel limit is earlier than the earliest deadline, the agent selects the task with the highest score. This condition is added on lines 9–11. A scenario with fuel constraints and no deadlines on tasks is used to further evaluate the performance of the proposed method in section 4.3.

4.3 Performance Analysis

In this section, the performance of the proposed rank-based conflict resolution is tested and compared using 3 different heuristics against the baseline BW-CBBA that is used as a benchmark. These different combinations are evaluated as a function of the number of iterations until convergence, the number of allocated tasks, and the distance travelled per task. A range of topologies is used to assess these performances since, as described in section 4.2.1, the topology affects the allocation dynamics. An increasing number of tasks with a fixed number of agents is also used to assess the performance of the algorithms ranging from when the system is under-constrained to over-constrained. Over-constrained signifies that there are a greater number of tasks than can be assigned given the time constraints, while under-constrained signifies that there is enough capacity to assign all tasks. A variation in the time constraints is also applied to further demonstrate the performance of the proposed method.

4.3.1 Assessing Performance

The combinations of the proposed rank-based conflict resolution with three different heuristics, and the benchmark algorithm, are detailed as follows:

1. **EDF-Rank**: Selecting tasks based on EDF (section 4.2.2) and rank-based conflict resolution (section 4.2.1) - (Algorithm 5).

2. **Score-Rank**: Selecting tasks based on score function (section 2.7) and rank-based conflict resolution (section 4.2.1). This configuration is Algorithm 5 with lines 8, 9 and 11 removed.

3. **Mixed-Rank**: Selecting tasks based on either EDF (section 4.2.2) or score function (section 2.7) and rank-based conflict resolution (section 4.2.1). This configuration applies EDF task selection to 4 agents and applies task selection based on scores to the other 10 agents.

4. **Score-Bids**: Selecting tasks based on score function and convergence with varying bids. This configuration is the benchmark Algorithm 3, first introduced in [45].

4.3.2 Experimental Setup

A simulated search and rescue scenario is used to test the performance of the algorithms, with a rescue team equally split into two agent types with different functions. The tested scenarios build on the environment types described in Chapter 3. One agent type provides medicine, the other provides food. The survivors are likewise equally split into those requiring food and those requiring medicine. The scenario specifications are summarised in Table 5.1. The mission takes place in a 3D space. The task locations are uniformly distributed within this 3D space, while the agents' starting positions are uniformly distributed on the 2D ground space. For these simulations, it is assumed that the agents are stationary during the task allocation. The deadlines for starting each rescue and the battery limits on each agent are uniformly distributed. Given the random initialisation of task and agent locations and deadlines, it is sometimes impossible for some tasks to be started by any agent before its deadline.

The reward and cost for the scoring function (equation 2.13) were set as $R = 10\,000$ and $C_{iql} = \Delta D_{iq}(\mathbf{p}_i)/vel_i$, where $\Delta D_{iq}(\mathbf{p}_i)$ is the distance travelled by the agent to reach the candidate task location from its previous location in $\mathbf{p}_i$, and vel_i is the velocity of agent i. The value for R was set to ensure that the score is greater than 0 after the cost is deducted.

The total iterations for one simulation is expressed as the last iteration number at which an allocation change was made, either through inclusion or removal. The travel distance

Table 4.1 Scenario Specification for Rank-Based Conflict Resolution

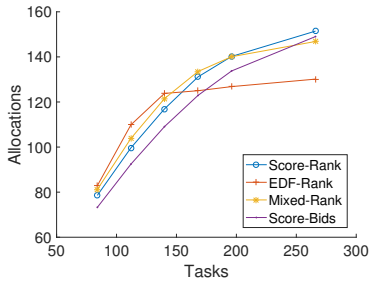
	Medicine	Food
Agent Speed	30m/s	50m/s
Agent Battery	Between 2500 and 5000 seconds	
Agent Start Position	10 000m x 10 000m x 0m ground space	
Task Duration	300 seconds	350 seconds
Task Deadline	Between 0 and 5000 seconds	
Task Location	10 000m x 10 000m x 1000m 3D space	

is represented as the average travelling distance per task for all agents with the final task allocation. The number of agents was fixed at 14 and the number of tasks tested was 84, 112, 140, 168, 196, and 266. These numbers were selected to cover a range from under-constrained to over-constrained. The increase in the number of tasks was arbitrarily selected. The number of agents 14 was selected as the largest number to fit within computer performance limitations. The agent network topologies used are illustrated in Fig. 5.1. The topology is initialised at the outset and remains constant through the task allocation process. Each setup was run 50 times with the same configuration but different initial conditions. Results are shown as averages over those 50 runs. All simulations were performed in MATLAB on an Intel(R) Xeon(R) CPU server with 2.2 GHz and 128Gb RAM on LINUX operating system Ubuntu 14.04.5 LTS

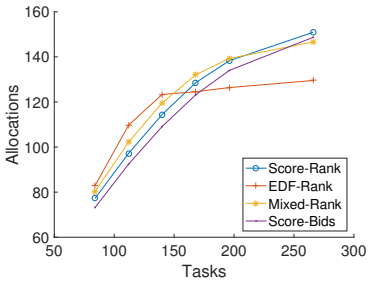
4.3.3 Results

Figure 4.2 plots the results of Score-Rank, EDF-Rank, Mixed-Rank and the benchmark Score-Bids across the different topologies as a function of the total number of tasks. The trend in the number of allocations is consistent across topologies. The algorithm using EDF allocates the highest average number of tasks for the lower 3 task numbers. In the best case, EDF-Rank allocates 17.4 more tasks on average than Score-Bids with ordered row topology. For all task numbers, Score-Rank allocates more tasks than Score-Bids. In the best case, Score-Rank allocates 8.2 tasks more on average than Score-Bids. Compared with EDF-Rank, The algorithms using Score allocate the most tasks for the highest 2 task numbers. A general trend is that EDF allocates the most tasks when the system is under-constrained i.e. the lower 3 task numbers. When the system is over-constrained, i.e. the higher 3 task numbers, Score allocates the most tasks by a clear margin.

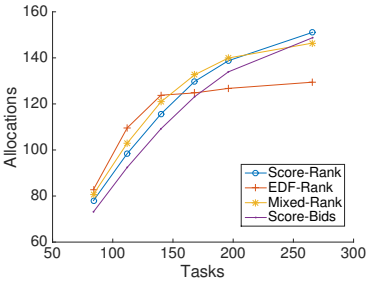
The performances of the algorithms using Rank consistently average at 7 iterations, with below 0.5 standard deviation, at all numbers of tasks with the ordered row topology. In comparison, the benchmark Score-Bids ranges from 13.5 to 17.5 average with between 4 and 5 standard deviation. The average number of iterations for the Rank algorithms increases with the unordered row topology, but remain lower than for Score-Bids. With the hybrid



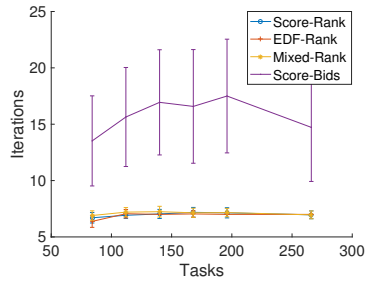
(a) Ordered Row Allocations



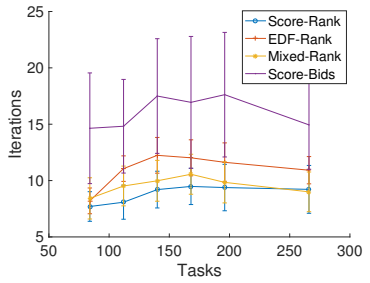
(b) Unordered Row Allocations



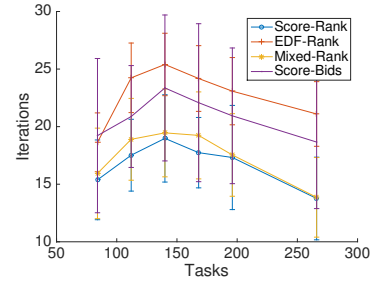
(c) Hybrid Allocations



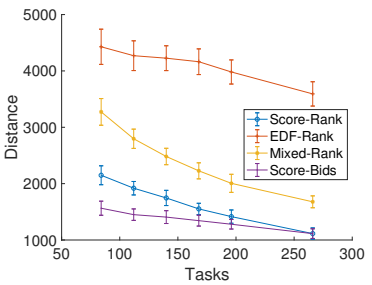
(d) Ordered Row Iterations



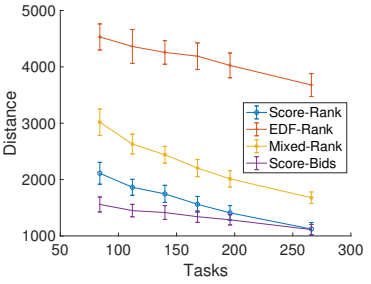
(e) Unordered Row Iterations



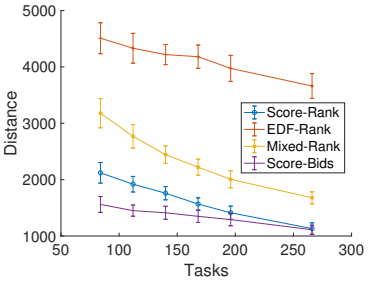
(f) Hybrid Iterations



(g) Ordered Row Travel Distance



(h) Unordered Row Travel Distance



(i) Hybrid Travel Distance

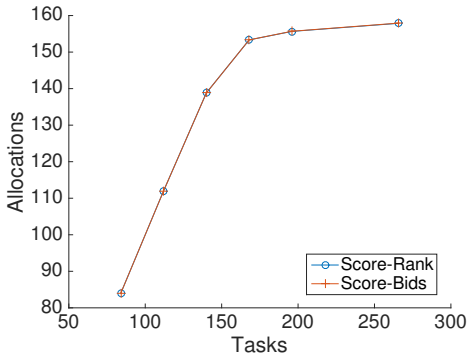
Fig. 4.2 Average allocations, iterations, and travel distance per task across different network topologies in a scenario with time constraints on tasks and on agents. The number of agents is 14 and the numbers of tasks are 84, 112, 140, 168, 196, and 266. The error bars represent standard deviation.

topology, Score-Rank consistently converges in fewer iterations on average than Score-Bids, whereas EDF-Rank converges slower on average than Score-Bids 5 out of 6 times.

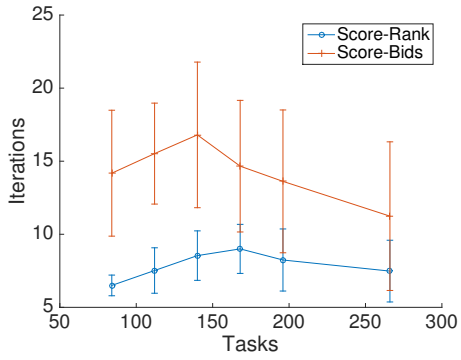
With the unordered row and hybrid topologies, Mixed-Rank achieves higher average allocations than the Score algorithms 5 out of 6 times. Similar results are achieved with the ordered row topology. With the unordered row and hybrid topologies, the corresponding average iterations for Mixed-Rank are second lowest. In the best case for the hybrid topology, Mixed-Rank allocates 11.7 tasks more than Score-Bids in 3.9 iterations on average fewer than Score-Bids.

The average travel distances per task are consistent across the three topologies. EDF-Rank gives the highest travel distance by a significant margin, between 3 and 4 times greater than Score-Bids, which achieves the lowest average distance. As might be expected, Mixed-Rank falls between EDF-Rank and Score-Rank proportionally to the split of agents using either heuristic. Interestingly, while Score-Rank gives higher average distances than Score-Bids, there remains a clear advantage towards optimising travel distances by using Score-Rank compared with EDF-Rank. With the higher numbers of tasks, there is not a significant difference in average travel distance between Score-Rank and Score-Bids. These results give an indication of the trade-off for speeding up convergence with a marginal increase in average travel distance, using the proposed method.

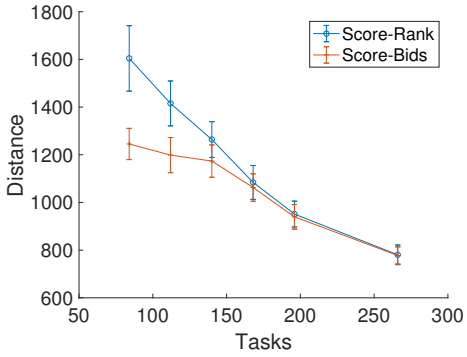
Figure 4.3 shows the results for the scenario with time constraints on agents without deadlines on tasks, using the unordered row topology. Score-Rank and Score-Bids are compared. The numbers of allocations consistently match for both algorithms. In the best case, Score-Rank converges in less than half the number of iterations compared with Score-Bids. The average travel distance per task is significantly higher with Score-Rank for the lower two task numbers. However, for the higher task numbers, the average travel distance is the same for both Score-Rank and Score-Bids.



(a) Allocations



(b) Iterations



(c) Travel Distance

Fig. 4.3 Average allocations, iterations, and travel distance per task for a scenario with fuel constraints on agents and without deadlines on tasks using the unordered row topology.

4.4 Discussion and Conclusions

Simulation results indicated that the proposed rank-based conflict resolution combined with insertion heuristics were successful for minimising time to convergence while maximising task allocations. The findings suggest that the proposed approach of rank-based conflict resolution is most effective and can strongly reduce convergence time when agents' ranks are determined by the network topology. Future work may look at a theoretical analysis of the proposed method to formally compare the average and worst case convergence times with previous methods. The proposed method may also be extended to assign agents' ranks based on the network topology. The performance of the proposed method may be further assessed under time-varying topologies. Results also showed that different insertion heuristics perform best in different environments. EDF allocated the most tasks when the number of tasks to agents was lower, while NTF allocated the most tasks when the number of tasks to agents was higher. These results motivate further studies to devise algorithms that can select the appropriate strategy autonomously. The strategy may be selected according to a dynamically changing number of tasks, number of agents and network connectivity links. In the next chapter, an approach is proposed that enables each agent to select the task allocation strategy which they independently predict is the best based on locally communicated bids.

Chapter 5

Autonomous Strategy Switching

5.1 Introduction

Current state-of-the-art consensus-based task allocation algorithms incorporate heuristics into agent score functions in order to optimise a given objective. While extensive research has been done in the area of multi-agent learning of optimal policies [76, 13], at the time of writing this, consensus-based task allocation algorithms have not been designed to adapt online to changing environmental factors [80, 44]. Results in the previous chapter indicated that different task allocation strategies perform most optimally depending on environmental factors such as the ratio of tasks to agents. This chapter investigates the hypothesis that each individual agent can predict and select the best task allocation strategy using information derived from local communications. This chapter introduces the novel idea of learning a prediction function and adopting a strategy switching behaviour that allows agents to independently adapt task allocation strategies in line with changing environmental factors, and boost performance. The learned function is effectively a prediction mechanism that uses past experience to select which task allocation strategy yields the optimal global task allocation.

The proposed method is tested through a simulated search and rescue scenario. Two heuristics that have been previously shown to perform well in such a scenario are earliest deadline first (EDF) and nearest task first (NTF) [68]. The prediction functions were trained

to predict which heuristic, between the two, will yield the most task allocations. The following assumptions are made: an agent does not have knowledge of the time availability of other agents in the system, and does not have knowledge of the decisions made by other agents concerning the optimal heuristic. The input for the prediction function is limited to information about task assignments received locally from networked agents. The reasoning for these choices is to show that the proposed adaptive method can be applied to consensus algorithms by exploiting the communications necessary for consensus, and without requiring any additional information to be communicated among agents. Results showed that for the majority of scenarios tested, the agents were able to predict and switch to the optimal heuristic based on observations of locally communicated task assignments, without a significant impact on the time to convergence. Additionally, results showed that an additional gain in performance could be achieved by enabling the agents to independently adapt their consensus strategy.

5.2 Learning Strategy Adaptation

This section introduces the proposed adaptive approach for consensus-based task allocation that enables agents to individually predict and select the best task inclusion strategy with the aim to automatically maximise task allocation.

5.2.1 Heuristic Strategies

In task allocation problems that require agents to execute multiple tasks, the heuristic with which agents include tasks into their schedules is key to optimising allocations. The appropriateness of any heuristic varies as conditions change, such as the number of tasks to agents, the time constraints, and the travel times between tasks. The two heuristics used in this study are earliest-deadline-first (EDF), and nearest-task-first (NTF) [68]. With EDF, agents prioritise tasks with the earliest deadline to include into their schedules, while with NTF, agents prioritise tasks that are nearest to the previous location in their schedule. In a standard environmental setting (see Section 5.2.6 for an example), EDF allocates more tasks

than NTF under conditions with relatively few tasks per agent. As the number of tasks per agent increases, the travel time between tasks becomes a greater factor and eventually NTF allocates the most tasks. As a result, traditional consensus-based task allocation algorithms perform often sub-optimally because there is not a single strategy that works well in all scenarios.

In this study, the key idea is that optimal strategies can be inferred online, i.e., during execution, and locally, i.e., in a distributed fashion for each agent. Thus, the information exchanged by the agents to reach consensus can be exploited to implement a distributed adaptive system. A decision-making mechanism is devised in which agents use the local information available to predict the best heuristics online. The process is shown to result result in the optimisation of the number of allocated tasks under a variety of different conditions.

To infer a rule connecting the local observations made by an agent and the appropriate heuristic, supervised classification learning is used. With supervised learning, the learning algorithm uses labeled training data to infer a general rule or function that maps inputs to outputs. In this study, the input is an agent's observation following a consensus phase, and the prediction is the heuristic that will yield the highest number of allocated tasks overall.

The proposed method exploits locally available task assignment information that is necessary for consensus. Agents are able to resolve conflicting assignments through sharing information about which agents are assigned to which tasks [17]. Thus, the proposed method can be integrated into consensus-based algorithms without additional communication overhead. The implementation of the proposed method is described as an extension to CBBA.

5.2.2 Agent Observations

As described in Chapter 2.7 that formalises CBBA, reaching consensus requires that agents exchange the list $\mathbf{z}$ of agent-task allocations. The list $\mathbf{z}_i$ corresponds to agent v_i's local knowledge of the current global task allocation. From these communications v_i can make the following local observations:

- The set of assigned tasks: $\mathbf{a} = \{k \in \mathbf{z}_i \mid \mathbf{z}_{ik} > 0\}$ and the set of unassigned tasks: $\bar{\mathbf{a}} = \{k \in \mathbf{z}_i \mid \mathbf{z}_{ik} = 0\}$. The cardinalities $|\mathbf{a}|$ and $|\bar{\mathbf{a}}|$ denote the total numbers of assigned tasks and unassigned tasks respectively.

- The set of tasks assigned to other agents not including tasks assigned to v_i is defined as: $\mathbf{o} = \{k \in \mathbf{z}_i \mid \mathbf{z}_{ik} > 0 \wedge \mathbf{z}_{ik} \neq i\}$, where $|\mathbf{o}|$ denotes the cardinality of $\mathbf{o}$.

When accounting for heterogeneous agents with different capabilities to perform different tasks, the observations refer to the tasks that v_i is capable of performing i.e. $|\mathbf{a}|$ denotes the number of compatible assigned tasks. We refer to the total number of compatible tasks as m_c. In summary, each agent can derive the following information from received communications: the number of assigned and unassigned tasks, the number of tasks assigned to other agents, and the total number of compatible tasks. This information is used to predict which allocation strategy is more likely to perform better as explained in the following sections. It is worth noting that additional information can be derived and used for predictions. We focus on the set described above as the most informative for the problem of interest, and allow for the possibility of extending the set of inputs in future work.

5.2.3 Learning Systems

The main focus of this study is the integration of learning and decision making into CBBA to demonstrate that, within the established framework of such an algorithm, appropriate predictions and decisions can be made. Thus, off-the-shelf supervised learning algorithms were used with default parameters to implement the prediction function. It is important to note that learning the prediction function is performed centrally and offline, while the adaptation of the strategy, using the learned function, is performed online and in a distributed fashion. Two popular supervised learning methods used in this study are: support vector machine (SVM) and neural network (NN). The proposed adaptive method using SVM and NN are referred to as CBBA$^{+}{}_{\text{SVM}}$ and CBBA$^{+}{}_{\text{NN}}$, respectively. The SVM model used a radial basis function kernel with the MATLAB function for binary classification *fitcsvm*. Using MATLAB R2017a's Neural Pattern Recognition toolbox, the two-layer feed-forward network

with sigmoid hidden and softmax output neurons was trained with default parameters using scaled conjugate gradient backpropagation and had a single hidden layer with 10 nodes. The inputs used for training corresponded to the observation: $[|\mathbf{o}|, |\bar{\mathbf{a}}|]/\ m_c$. Two outputs corresponded to the classification predictions of which strategy (between EDF and NTF) led to the most tasks being allocated in previous task allocation experiments with non-adaptive strategies. The SVM model returns 0 or 1 corresponding to the classification prediction of an observation. The neural network returns a real number between 0 and 1, indicating the confidence in the classification prediction, where an output of 1 indicates the highest confidence in the classification, and an output of 0 indicates the lowest confidence in the classification.

5.2.4 Distributed Strategy Adaptation

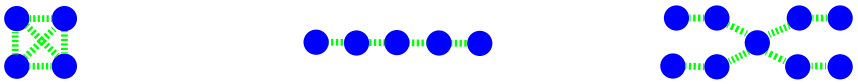
The task allocation algorithm with the added prediction function is shown with pseudocode in algorithm 6. Before the task allocation procedure begins, each agent is initialised with an index h that determines with which heuristic the agent includes tasks into its schedule (line 3). This initialisation can be done through a uniform random assignment. Once the task allocation procedure is in progress, predictions can be made using locally received information about task assignments. The Predict function (line 5) uses $\mathbf{z}_i$ to make a prediction on the optimal heuristic and returns a heuristic index h. This index is passed to the Task Inclusion Phase (line 6) where v_i includes tasks into its schedule according to the heuristic corresponding to h. During the consensus phase (line 7), $\mathbf{z}_i$ is updated.

The prediction function is shown with pseudocode in algorithm 7. Applying a limit to the number of times that an agent can switch functions is fundamental to maintain the guarantee of convergence of the algorithm (see [17] for details on convergence). Therefore, a condition $SwitchCondition$ on line 3 determines whether a prediction can be made and therefore whether the heuristic can be changed or not. In this study, that limit is set to 1 to test the basic concept that one single switch of strategy is sufficient to increase the overall number of allocated tasks. For a real-time system operating in a dynamic environment, in which agents converge locally rather than as a group [46], a refractory period could be

implemented to allow for multiple switches over time with a delay in between. To ensure that the agent has sufficient information to make a prediction, the *SwitchCondition* applied in this study requires that the agent has received task assignment information from other agents, such that: $|\mathbf{o}| > 0$. If the condition returns false then the agent's current heuristic index is returned. Before computing a prediction, the input for the prediction function is normalised by dividing it by the number of tasks: $input = [|\mathbf{o}|, |\bar{\mathbf{a}}|]/m_c$ (line 4). The output computed by the prediction function $f_{predict}$ is evaluated to determine which heuristic is likely to generate the optimal task allocation. If the predicted heuristic is different from the agent's current heuristic, then the agent unassigns all tasks previously assigned to itself so that it can rebuild its schedule with the predicted optimal heuristic (lines 7-11). CBBA's task inclusion phase (see [17]) is the algorithm's point of highest time complexity, consisting of three nested loops. As the proposed adaptive strategy function runs outside of the task inclusion phase and does not require loops, the algorithm's time complexity is unaffected.

Agents communicate task allocations according to a network topology, which impacts the task assignment information that an agent holds at any given time. Moreover, agents bidding on the same task types may be connected through multiple agents bidding on different task types. Therefore, it may take several rounds of the algorithm before an agent receives bid information from same-type agents, which is the key information used for the prediction of the best strategy. Different topologies result in different delays to the agents receiving sufficient information for making an accurate prediction. Early predictions may therefore benefit from being delayed until more task assignment information is gathered from the rest of the network. Figure 5.1 illustrates examples of common topologies. With a fully connected topology (Figure 5.1(a)), each agent receives information about all other agents' task assignments at every communication round. With a row topology (Figure 5.1(b)), it may take many rounds of CBBA for an agent's allocations to be propagated through the network.

Considering the possible delays in receiving sufficient information, and the requirement to limit the number of times an agent switches heuristic, we apply basic rules to ensure that the adaptive system is able to work under such constraints. After the training phase, the NN gives an output of 0.5 when no strategy has a clear advantage over the other. A



(a) Fully connected (b) Row connected (c) Star connected

Fig. 5.1 Network topologies: agents are represented as circles and communication connections are represented as dashed lines between agents. Different topologies affect the timing at which each agent acquires information on allocations. Thus, the decision making capabilities of the agents may be affected by different topologies.

question is, how much more advantageous a strategy needs to be to trigger a switch? Given the incompleteness of the information available to agents through local communication, it is reasonable to assume that a strategy prediction requires a confidence margin to signal that switching is advantageous. A ROC (receiver operating characteristic) curve shows graphically the true positive rate as a function of the false positive rate for different cut-off points. A ROC analysis therefore can provide experimental evidence to estimate a confidence parameter so that switching occurs with a desired probability. In the proof-of-concept presented in this study, the threshold to switch for $f_{evaluate}$ in CBBA$^{+}_{\text{NN}}$ is set to an arbitrary value of 0.6. This simple adjustment prevents strategy switching when no clear advantage for one strategy can be inferred. This prevents unnecessary strategy switching and maintains a low number of iterations as shown in the analysis later. Further studies could address the tuning of such a parameter to achieve the best compromise between reactive switches and number of iterations to convergence. In other words, introducing this condition effectively results in the agent delaying a decision until there is a sufficient confidence in either heuristic. Similarly, the decision system is adapted to operate with predictions from the SVM: in this case, CBBA$^{+}_{\text{SVM}}$ agents can perform a switch only when $T > 5$, so that each agent has received task allocation information from multiple other agents before switching heuristic in the worst case topology tested. These mechanisms highlight the important fact that decision making in consensus-based algorithms cannot be simply left to a prediction function, but needs to take into consideration the collective multi-agent dynamics. Future studies may investigate further the tuning and implications of different decision making rules.

A factor that affects allocations in consensus-based algorithms is the conflict resolution mechanism. The most common approach to resolving conflicts in consensus task allocation

algorithms is to assign tasks to the highest bidder. This process can either happen via an auctioneer [25], or can be fully distributed as with CBBA [17]. Variations of this process exist to account for different problem constraints [18, 23, 10, 20]. A second mechanism consists of utilising relative ranking among agents [96]. To assess how well the proposed approach generalises with different conflict resolution strategies, both the bid-based and the rank-based conflict resolution procedures are tested. The implementation of rank-based conflict resolution is easily performed thanks to the tie-breaking heuristic based on agents' unique identification numbers [17] built into CBBA. If all agents place bids of the same value, all conflicts are resolved based on the agents' IDs, which can be thought of as the

Algorithm 6 Task allocation outer-loop iterative procedure with predictive function running on v_i

initialise timer $T \leftarrow 1$
$converged \leftarrow false$
initialise h
while *converged* is $false$ **do**
    h = Predict(h,$\mathbf{z}_i$)
    TaskInclusionPhase(h)
    Consensus Phase
    converged $\leftarrow$ Check Convergence.
    $T \leftarrow T + 1$
end while

Algorithm 7 Prediction function for optimal task inclusion strategy running on v_i

function PREDICT(h_{curr},$\mathbf{z}_i$)
    Compute $|\mathbf{o}|$,$|\bar{\mathbf{a}}|$ from $\mathbf{z}_i$ // Number of tasks assigned to others and unassigned
    if *SwitchCondition* is $true$ **then**
        $input = [|\mathbf{o}|, |\bar{\mathbf{a}}|]/m_c$
        $output = f_{predict}(input)$
        $h_{new} = f_{evaluate}(output)$
        if $(h_{curr} \neq h_{new})$ **then** // If predicted strategy is different from current strategy
            Empty $\mathbf{p}_i$ // Empty task list
            Set all $\mathbf{z}_{ik} = i$ to $\mathbf{z}_{ik} = 0$ // Reset own bids
            $h_{curr} = h_{new}$ // Update strategy
        end if
    end if
    return h_{curr}
end function

agents' rank. Agents place all bids equal to the constant *MaxBid* for Rank-based conflict resolution.

5.2.5 Benchmark Algorithms

The proposed adaptive approach with $\text{CBBA}^{+}_{\text{NN}}$ and $\text{CBBA}^{+}_{\text{SVM}}$ is compared to variations of the non-adaptive baseline CBBA:

- CBBA_{EDF} - all agents use EDF.
- CBBA_{NTF} - all agents use NTF.
- $\text{CBBA}_{50/50}$ - half of the agents use EDF and half use NTF.

CBBA resolves conflicting task assignments by assigning tasks to the highest bidder. For each of these algorithms, agents place bids to the value determined by the score function using NTF. Thus, conflicting task assignments are resolved based on which agent can reach the task fastest from the previous location in their schedule.

5.2.6 Preparation of Dataset

A simulated search and rescue scenario is used to test the performance of the algorithms, with a rescue team equally split into two agent types with different functions. One agent type provides medicine, the other provides food. The survivors are likewise equally split into those requiring food and those requiring medicine. The task allocations for these two job types are solved independently, but require agents of both types to contribute in message passing and to resolve conflicts. The scenario specifications are summarised in Table 5.1. The task locations are uniformly distributed within a 3D space, while the agents' starting positions are uniformly distributed on the 2D ground space. The deadlines for starting each rescue and the battery limits for each agent are uniformly distributed. Given the random initialisation of task and agent locations and deadlines, it is sometimes impossible for some tasks to be started by any agent before its deadline.

The training set is generated by running task allocation experiments under various configurations. The task and agent numbers were selected to cover a range from under-constrained to over-constrained. Over-constrained signifies that there are a greater number of tasks than can be assigned given the time constraints, while under-constrained signifies that there is enough capacity to assign all tasks. Each observation was labeled corresponding to whether CBBA_{EDF} or CBBA_{NTF} yielded the highest number of allocated tasks overall at the time of convergence. Under a star communication network topology, the number of agents was fixed at: 14, and the numbers of tasks were: 84, 112, 140, 168, 196, 266. Under a fully connected communication network topology, the numbers of agents were: 4, 6, 8, 10, 12, 14, 16 and the number of tasks was fixed at: 130. To add variation, this latter setup was repeated with both agent types able to service both task types. The increase in number of tasks and agents were arbitrarily selected within a range to cover a variety of tasks to agent ratios, from under-constrained to over-constrained. Each setup was run 50 times with the same configuration but different initial conditions.

From simulations running these configurations with CBBA_{EDF} and CBBA_{NTF}, the observations: $[|\mathbf{o}|, |\bar{\mathbf{a}}|]/\, m_c$, were taken from each agent at each iteration starting from $T = 2$ to the time of convergence. Given the high number of agents deployed in one scenario and the repetition of scenarios, input vectors $[|\mathbf{o}|, |\bar{\mathbf{a}}|]/\, m_c$ with identical values and labels may be observed in the dataset. Such data points are effectively duplicates and can be safely removed from the dataset. After removal of duplicates, the labeled data set consisted of approximately 6000 unique observations. Cases for which the two heuristics were equivalent were left in.

Table 5.1 Scenario Specification for Adaptive CBBA

	Medicine	Food
Agent Speed	30m/s	50m/s
Agent Battery	Between 2500 and 5000 seconds	
Agent Start Position	10 000m x 10 000m x 0m ground space	
Task Duration	300 seconds	350 seconds
Task Deadline	Between 0 and 5000 seconds	
Task Location	10 000m x 10 000m x 1000m 3D space	

5.3 Performance Analysis

The simulation results compare the performances of the different algorithms with respect to average task allocations and iterations until convergence at the end of the task allocation process. In real-time systems, the time to reach a solution may be critical to successfully completing the mission. Thus, our analysis also investigates whether strategy adaptation allows the system to converge to a solution within similar time to non-adaptive algorithms. The total iterations for one simulation is determined by the last time an allocation change was made by any agent, either through inclusion or removal. A marginal increase in average execution time per iteration is expected with the adaptive strategies compared to the non-adaptive algorithms. In real-time settings, variable factors that depend on the specific implementation, such as the time required for communication, the processing speed, the number of tasks, the number of times the agent attempts to make a prediction, are all factors that may impact the proportional increase in average execution time. These points are worth investigating in future work to evaluate the trade-off. Results are shown as averages over 50 runs. All simulations were performed in MATLAB on an Intel(R) Xeon(R) CPU server with 2.2 GHz and 128Gb RAM on LINUX operating system Ubuntu 14.04.5 LTS.

5.3.1 Unseen Row Topology, Task Numbers, and Rank-Based Conflict Resolution

This section shows the results of tests comparing the algorithms operating with 14 agents under conditions not seen in training: under a row topology, with different task numbers, and a different conflict resolution strategy. In Figure 5.2(a), the proposed $\text{CBBA}^{+}_{\text{NN}}$ and $\text{CBBA}^{+}_{\text{SVM}}$ both match the best average numbers of allocations achieved by the non-adaptive approaches showing that the agents are correctly predicting and selecting the optimal heuristic under different conditions. The number of iterations until convergence are similar for the proposed adaptive approach and the non-adaptive approach that the agents are selecting, indicating that strategy adaptation maintains a similarly low number of iterations as the non-adaptive cases. $\text{CBBA}^{+}_{\text{NN}}$ takes marginally longer to converge on average than CBBA_{EDF}

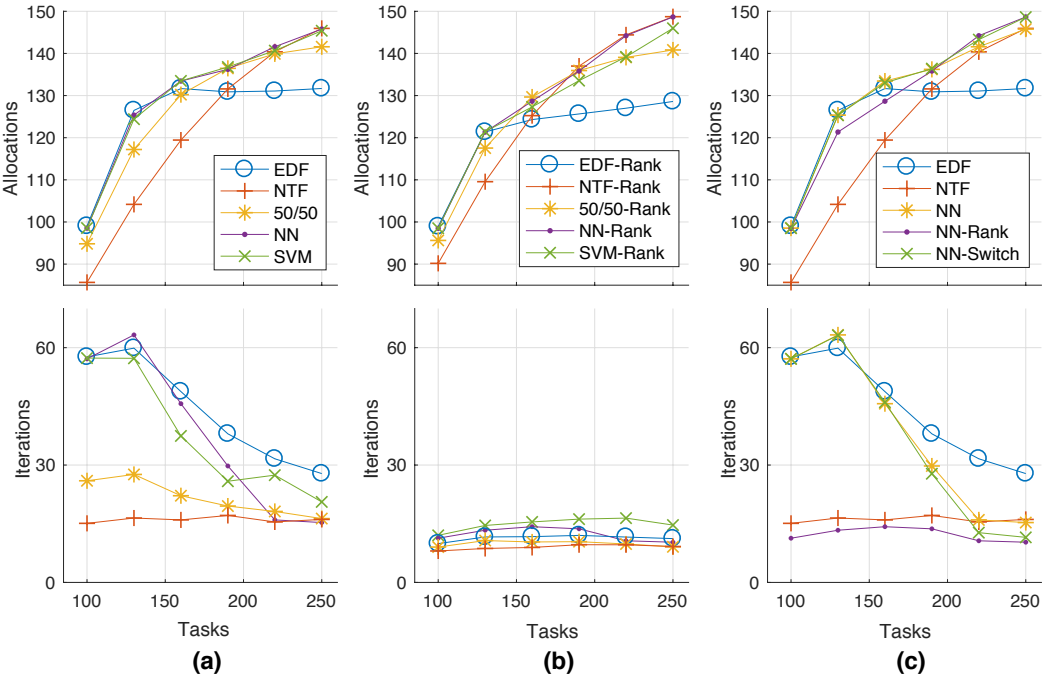


Fig. 5.2 Average task allocations (top) and average iterations until consensus (bottom) for scenarios with different task numbers (100,130,160,190,220,250) and a fixed number of networked agents (14), connected with a row topology. In a) five algorithms are compared: all agents self-assign tasks with the earliest-deadline-first (EDF) heuristic; all agents self-assign tasks with the nearest-task-first (NTF) heuristic, agents are split half and half into using EDF and NTF respectively (50/50); agents are initialised with 50/50 and then optionally switch to EDF or NTF based on a trained neural network (NN) prediction; agents are initialised with 50/50 and then optionally switch to EDF or NTF based on a support vector machine (SVM) prediction. In b) the same algorithms resolve conflicts according to the relative ranking of agents (Rank). In c) with NN-Switch, the task inclusion is as with NN, and conflict resolution switches to Rank if the optimal task inclusion heuristic is predicted to be NTF.

and CBBA$^{+}_{\text{SVM}}$ at 130 tasks, but matches the fastest convergence time of CBBA$_{\text{NTF}}$ at 220 and 250 tasks. CBBA$^{+}_{\text{SVM}}$ is relatively faster to converge for the lower task numbers and relatively slower for the higher task numbers compared with CBBA$^{+}_{\text{NN}}$.

Figure 5.2(b) shows the results with all algorithms using the Rank-based conflict resolution strategy, where agents resolve conflicts on task assignments based on agents' ranks. CBBA$^{+}_{\text{NN}}$-Rank still matches the best allocation numbers compared with the non-adaptive approaches, mostly unaffected that Rank consensus was not seen during training. CBBA$^{+}_{\text{SVM}}$-Rank matches the best average numbers of allocations for the lower task numbers, but for the higher numbers shows a drop in performance compared with CBBA$^{+}_{\text{NN}}$-Rank. However, CBBA$^{+}_{\text{SVM}}$-Rank still allocates more tasks on average than CBBA$_{\text{EDF}}$-Rank. The time to convergence for each algorithm is faster overall with Rank consensus, and average time taken is comparable for each algorithm. CBBA$^{+}_{\text{NN}}$-Rank takes at most 2 extra iterations on average than the slowest non-adaptive algorithm, and at best 1 iteration less. CBBA$^{+}_{\text{SVM}}$-Rank is the slowest to converge.

Figure 5.2(c) shows that when agents use the NTF heuristic in the scenarios with the higher numbers of tasks, the agents allocate more tasks overall on average if combined with Rank-based conflict resolution. The experiments for CBBA$^{+}_{\text{NN}}$ were repeated with the added condition that if an agent predicts that NTF is the optimal heuristic, it also switches to using Rank-based conflict resolution. The results are plotted as NN-Switch. Figure 5.2(c) shows that for the higher number of tasks, NN-Switch benefits from the higher allocations enabled by Rank-based conflict resolution, as well as the faster convergence time compared with CBBA$^{+}_{\text{NN}}$ and CBBA$_{\text{NTF}}$. For the higher number of tasks, the convergence time for NN-Switch is closest to CBBA$^{+}_{\text{NN}}$-Rank, which has the fastest convergence. In the lower task numbers, NN-Switch benefits from the higher allocations afforded by using bids for conflict resolution, and matches the slower convergence times of CBBA$^{+}_{\text{NN}}$ and CBBA$^{+}_{\text{EDF}}$. In these scenarios, the task inclusion strategy and the consensus strategy both affect the performance of the task allocation algorithm.

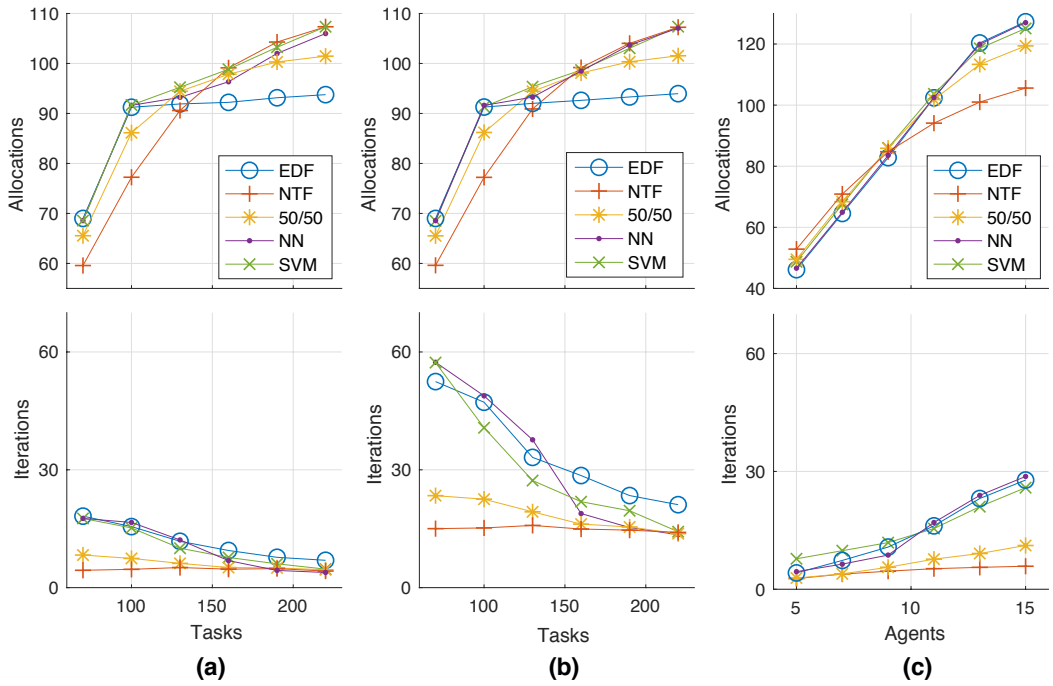


Fig. 5.3 Average task allocations (top) and average iterations until consensus (bottom). In a) and b) task numbers are (70,100,130,160,190,220) with a fixed number of agents (10). In a) agents are connected with a fully connected topology, in b) a star topology. In c) a fixed number of tasks (130), agent numbers are (5,7,9,11,13,15) with a fully connected topology. The algorithms using EDF, NTF, 50/50, NN, and SVM are compared.

5.3.2 Unseen Agent Numbers and Task Numbers

This section shows the results of tests comparing the algorithms operating with different and varying numbers of agents, as well as task numbers unseen in training. Figure 5.3(a) and Figure 5.3(b) plot results with a fixed number of agents (10) and different numbers of tasks unseen in training. In Figure 5.3(a) the topology is fully connected and in Figure 5.3(b) it is star connected. Under both the full and star topologies, $\text{CBBA}^{+}_{\text{SVM}}$ is able to consistently match the average allocations achieved by the best non-adaptive approaches in a comparable convergence time. $\text{CBBA}^{+}_{\text{NN}}$ performs marginally less well in this scenario compared with $\text{CBBA}^{+}_{\text{SVM}}$. With the fully connected topology, $\text{CBBA}^{+}_{\text{NN}}$ falls short of achieving the highest average allocations for 4 out of the 6 task numbers. With the star topology, $\text{CBBA}^{+}_{\text{NN}}$ only falls short once when the best non-adaptive algorithm is $\text{CBBA}_{50/50}$, which

is not used for training. The convergence times of the proposed adaptive approaches are again comparable to the non-adaptive baseline approaches.

Figure 5.3(c) plots results for simulations with different unseen agent numbers and a fixed number of tasks (130) under a fully connected network. With the higher number of agents (11,13, and 15), $\text{CBBA}^{+}_{\text{NN}}$ and $\text{CBBA}^{+}_{\text{SVM}}$ perform well in allocating tasks by accurately predicting and selecting the optimal heuristic. The proposed adaptive algorithms perform less well for the lower number of agents (5 and 7). $\text{CBBA}^{+}_{\text{SVM}}$ matches $\text{CBBA}_{50/50}$ for number of allocations which achieves the second highest average allocations of the non-adaptive approaches, while $\text{CBBA}^{+}_{\text{NN}}$ predicts incorrectly that EDF is the optimal heuristic. With 5 and 7 agents, $\text{CBBA}^{+}_{\text{NN}}$ and $\text{CBBA}^{+}_{\text{SVM}}$ also converge marginally slower than the non-adaptive approaches.

It is worth noting that $\text{CBBA}_{50/50}$ performs well in all the tested scenarios and offers good convergence speed. Adjusting the ratio to have more agents using EDF proportionally increases the average number of allocations for the lower task numbers, and reduces the average number of allocations for the higher task numbers. The inverse holds true when the ratio favours agents using NTF. For simple problems, this static approach is a viable alternative to the proposed approach. The proposed adaptive approach instead offers a proof of concept that can be extended for more complex scenarios. In fact, more sophisticated heuristics can be added or learned to give agents greater adaptability and ability to optimise the task allocation. Such an increase in flexibility and ability to optimise the task allocation would justify the use of the proposed adaptive approach compared to a static approach.

5.4 Conclusions

This study investigated the performance gain of enabling distributed agents to independently adapt their task allocation strategies according to locally received information. An adaptive distributed approach is proposed that combines a prediction function with a decision making capability to select the predicted optimal strategy. Results showed that in the majority of scenarios tested, a performance gain was achieved by using the proposed approach. Agents

were able to predict and select the optimal task inclusion heuristic to optimise the number of allocated tasks. In a minority of cases tested, when the number of agents was lowest, the agents predicted the incorrect heuristic. However, this resulted in a performance no worse than the non-adaptive strategy. Preliminary results showed that agents could further optimise the task allocation by adapting their conflict resolution strategy.

Factors such as the training data, the inputs, the machine learning tool, and the time of the prediction, are all factors that may impact the accuracy of the predictions, and are therefore interesting points to consider more deeply in future work. The proposed method could be extended to support a greater number of heuristics. For problems of greater complexity, additional inputs could be tested for increased accuracy. Additional inputs may include the number of tasks an agent removes during a round due to conflicts. Furthermore, the proposed approach could be adapted to support agents in learning the best strategy online, as well as adapting to changing optimisation objectives.

Chapter 6

Conclusions

This thesis introduced techniques for distributed task allocation algorithms to boost the task allocation solution. In particular, this thesis addressed the problem of maximising the number of task allocations in scenarios with time constraints, minimising the time to convergence, and augmenting agents with the capability to adapt their strategies to better meet these objectives. A bidding scheme was introduced that enables agents to reassign tasks such as to create feasible schedule space for unassigned tasks. A consensus policy based exclusively on agent ranking was introduced that speeds up the time to convergence. An adaptive approach using a prediction function and a decision making capability was introduced to enable agents to adapt their task allocation strategies in line with changing factors. In this chapter, the main contributions of the thesis are summarised.

6.1 Summary of Contributions

1. The main contribution described in Chapter 3 was the introduction of the algorithm PI-MaxAss, an extension of PI. The proposed PI-MaxAss uses a bidding scheme designed for increasing the number of allocated tasks in scenarios that prevent all tasks from being assigned. This novel scheme allows for simple and efficient reassignment of allocated tasks that enables the allocation of additional tasks. The method allows for task reassignment chains that can be limited to a predetermined maximum number of

agents. Simulations showed a noteworthy increase in the total number of allocated tasks and confirmed that the proposed algorithm can be applied beneficially to an existing scheduling method, thus opening the possibility of integration to other implementations. Maximising the number of allocated tasks is an important problem in scenarios such as search and rescue. The proposed method is most applicable to scenarios in which the ratio of tasks to agents is relatively low, where the decision of which agent gets which task is more significant to the overall task allocation than the order in which agents plan to execute their tasks. Due to the higher convergence time of PI, this method is also recommended for applications in which the time to convergence is not a significant factor.

2. The contribution introduced in Chapter 4 is a bidding scheme based exclusively on the relative rank of agents that aims to speed up the rate of convergence, without compromising the number of task allocations. The method is proposed as an extension to CBBA and incorporates well known insertion heuristics. Simulation results showed that the proposed rank-based conflict resolution combined with insertion heuristics proved successful for reducing the time to convergence and increasing the number of task allocations compared to a benchmark. The findings suggest that the proposed approach to resolve conflicts based on agents' ranks is most effective and can strongly reduce convergence time when agents' ranks are determined by the network topology. This approach would be most beneficial for teams of agents whose topology changes infrequently relative to the time required to reach consensus. Another result in this study is that faster consensus can be effectively achieved by employing multiple selection strategies across agents. The rate of convergence is an important factor in real-time systems, as time taken up by planning is time not spent on execution. The novel approach proposed here may enable real-world task allocation algorithms with slow or unreliable communication to reduce the number of messages exchanged by the robots and thus significantly reduce the time to convergence.

3. The contribution described in Chapter 5 introduces a distributed prediction mechanism that learns from past experience to enable an agent to select the task allocation strategy that yields the optimal global task allocation. The proposed adaptive method exploits the communications necessary for the agents to reach consensus, without requiring any additional information to be communicated among agents. Results showed that for the majority of scenarios tested, the agents were able to predict and switch to the optimal heuristic based on observations of locally communicated task assignments, without a significant impact on the time to convergence. Additionally, results showed that an additional gain in performance could be achieved by enabling the agents to independently adapt their consensus strategy. In dynamic real-time environments, there are many unknown changing factors that can affect the relative performance of any heuristic. Therefore, the performance of static approaches may suffer due to the changeability of the environment. This study aimed to offer a proof of concept to demonstrate the potential performance gain of enabling distributed agents to independently adapt their task allocation strategies according to locally received information. This simple approach can be extended for more complex scenarios with more sophisticated heuristics that would enhance agents' adaptability and capability to optimise the task allocation.

6.2 Future Research directions

The work in this thesis proposes optimisation techniques for distributed task allocation algorithms that have been tested in simulation. This section details some promising directions to extend the research.

1. Chapter 3 demonstrated the optimisation of two objectives with a two-step procedure that optimises one objective at a time through the tuning of score and bid functions. Future work could investigate the possibility of optimising additional objectives through additional steps. One of the drawbacks of changing bid functions when bids are reflective of the optimality of an assignment is that the bids lose meaning if two agents are attempting to optimise different objectives. In this case, as was the setup in Chapter 3,

the team of networked agents is required to coordinate the time at which they switch objectives. It would be interesting to investigate a bidding scheme that preserves the information of the optimality of a task assignment for different objectives, such as having the agents communicate multiple bid values for a single task assignment, and combine this with a mechanism similar to the adaptive approach described in Chapter 5. This can enable agents to switch objectives autonomously and asynchronously according to a prediction of which objective needs to be prioritised. It would also be worth comparing the performance of such an approach to a static method with a bidding scheme designed a priori to incorporate multiple objectives, which does not have the ability to tune which objective is prioritised online.

2. The performance of the fast convergence technique introduced in Chapter 4 is most effective when agents' ranks are determined by the network topology. In future work, it would therefore be interesting to look into combining the rank-based conflict resolution approach with a fast, flexible and robust method to assign ranks to agents based on the network topology. The potential in terms of performance could hypothetically achieve consensus in linear or close to linear time complexity, and as demonstrated by the results in Chapter 4, maintain a high number of allocations. In combination with such an extension, it would be beneficial to analyse and prove theoretically the convergence properties of the proposed method, and provide a formal comparison of the average and worst case convergence times as compared with previous methods. A hypothesis of what contributes to the effectiveness of the rank-based conflict resolution posits that when the ratio of tasks to agents is high, the strategy with which agents self assign tasks is more significant to the overall task allocation than how agents resolve conflicts among each other. A theoretical analysis of the conflict resolution strategy will produce better guarantees of convergence. In terms of application, more sophisticated insertion heuristics could be employed by the agents to optimise the optimality of their decisions when self-assigning tasks. The performance could then be analysed to determine if the initial trade-off incurred by losing bid information can be further compensated by more advanced insertion heuristics.

3. The proposed adaptive approach described in Chapter 5 has promising research directions. Optimising task allocations in dynamic and changing environments for real-time systems is an important problem. The ability of agents to autonomously and efficiently adapt their task allocation strategies in line with changing environmental factors is an exciting research avenue. For the proposed method as applied to consensus-based task allocation algorithms, future research could investigate a wider range of task allocation problems, such as those with cross-schedule dependencies, in combination with a comparison of the accuracy of the prediction functions given different sized training data. The training data, the inputs, the machine learning tool, and the time of the prediction, are all factors that can impact the accuracy of the prediction functions, therefore a deeper investigation into these parameters would provide a useful reference for the design of adaptive consensus-based task allocation systems. For problems of greater complexity, additional inputs could be tested to determine the effect on accuracy. Additional inputs may include the number of tasks an agent removes during a round due to conflicts. The proposed method could be extended to support more advanced heuristics, this would enhance agent's capability to optimise the global task allocation objective. Furthermore, the proposed approach could be adapted to support agents in learning the best strategy online, as well as adapting to changing optimisation objectives.

References

[1] Sergey Alatartsev, Sebastian Stellmacher, and Frank Ortmeier. 2015. Robotic Task Sequencing Problem: A Survey. *Journal of Intelligent & Robotic Systems* 80, 2 (2015), 279–298. https://doi.org/10.1007/s10846-015-0190-6

[2] Christopher Amato, Girish Chowdhary, Alborz Geramifard, N. Kemal Ure, and Mykel J. Kochenderfer. 2013. Decentralized control of partially observable Markov decision processes. In *52nd Annual Conference on Decision and Control (CDC)*. IEEE, 2398–2405.

[3] Christopher Amato, George Konidaris, Gabriel Cruz, Christopher A. Maynor, Jonathan P. How, and Leslie P. Kaelbling. 2015. Planning for decentralized control of multiple robots under uncertainty. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 1241–1248.

[4] Claudia Archetti, Mathieu Bouchard, and Guy Desaulniers. 2011. Enhanced branch and price and cut for vehicle routing with split deliveries and time windows. *Transportation Science* 45, 3 (2011), 285–298.

[5] Jonathan F. Bard, George Kontoravdis, and Gang Yu. 2002. A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science* 36, 2 (2002), 250–269.

[6] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W.P. Savelsbergh, and Pamela H. Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. *Operations research* 46, 3 (1998), 316–329.

[7] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research* 27, 4 (2002), 819–840.

[8] Andrea Bettinelli, Alberto Ceselli, and Giovanni Righini. 2011. A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies* 19, 5 (2011), 723–740.

[9] Giulio Binetti, David Naso, and Biagio Turchiano. 2012. Decentralized task allocation for heterogeneous agent systems with constraints on agent capacity and critical tasks. (2012), 1627–1632. https://doi.org/10.1109/ROBIO.2012.6491200

[10] Giulio Binetti, David Naso, and Biagio Turchiano. 2013. Decentralized task allocation for surveillance systems with critical tasks. *Robotics and Autonomous Systems* 61, 12 (2013), 1653–1664. https://doi.org/10.1016/j.robot.2013.06.007

[11] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. 1999. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press.

[12] Arne Brutschy, Giovanni Pini, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo. 2014. Self-organized task allocation to sequentially interdependent tasks in swarm robotics. *Autonomous agents and multi-agent systems* 28, 1 (2014), 101–125.

[13] Lucian Busoniu, Robert Babuska, and Bart De Schutter. 2008. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38, 2 (2008), 156–172. https://doi.org/10.1109/TSMCC.2007.913919

[14] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. 2013. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial Informatics* 9, 1 (2013), 427–438. https://doi.org/10.1109/TII.2012.2219061

[15] Jesus Cerquides, Alessandro Farinelli, Pedro Meseguer, and Sarvapali D. Ramchurn. 2013. A tutorial on optimization for multi-agent systems. *Comput. J.* 57, 6 (2013), 799–824.

[16] Archie C. Chapman, Alex Rogers, Nicholas R. Jennings, and David S. Leslie. 2011. A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems. *The Knowledge Engineering Review* 26, 4 (2011), 411–444.

[17] Han-Lim Choi, Luc Brunet, and Jonathan P. How. 2009. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics* 25, 4 (2009), 912–926.

[18] Han-Lim Choi, Andrew K. Whitten, and Jonathan P. How. 2010. Decentralized task allocation for heterogeneous teams with cooperation constraints. In *American Control Conference (ACC)*. IEEE, 3057–3062.

[19] Jens Clausen. 1999. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen* (1999), 1–30.

[20] Rongxin Cui, Ji Guo, and Bo Gao. 2013. Game theory-based negotiation for multiple robots task allocation. *Robotica* 31, 6 (2013), 923–934. https://doi.org/10.1017/S0263574713000192

[21] Rina Dechter and David Cohen. 2003. *Constraint processing*. Morgan Kaufmann.

[22] Erik Demeulemeester and Willy Herroelen. 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management science* 38, 12 (1992), 1803–1818.

[23] Donato Di Paola, Andrea Gasparri, David Naso, and Frank L. Lewis. 2014. Decentralized dynamic task planning for heterogeneous robotic networks. *Autonomous Robots* 38 (2014), 31–48. https://doi.org/10.1007/s10514-014-9395-y

[24] Donato Di Paola, David Naso, and Biagio Turchiano. 2011. Consensus-based robust decentralized task assignment for heterogeneous robot networks. In *American Control Conference (ACC)*. IEEE, 4711–4716.

[25] M. Bernardine Dias, R. Zlot, N. Kalra, and A. Stentz. 2006. Market-Based Multirobot Coordination: A Survey and Analysis. *Proc. IEEE* 94, 7 (2006), 1257–1270. https://doi.org/10.1109/JPROC.2006.876939

[26] Jittat Fakcharoenphol, Chris Harrelson, and Satish Rao. 2007. K-Traveling Repairmen Problem. *ACM Transactions on Algorithms* 3, 4 (2007), 40. https://doi.org/10.1145/1290672.1290677

[27] Alessandro Farinelli, Luca Iocchi, and Daniele Nardi. 2004. Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34, 5 (2004), 2015–2028.

[28] Alessandro Farinelli, Alex Rogers, and Nick R. Jennings. 2014. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Autonomous agents and multi-agent systems* 28, 3 (2014), 337–380.

[29] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R. Jennings. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *7th international joint conference on Autonomous agents and multiagent systems*, Vol. 2. IFAAMAS, 639–646.

[30] Hamed Fazlollahtabar, Mohammad Saidi-Mehrabad, and Jaydeep Balakrishnan. 2015. Mathematical optimization for earliness/tardiness minimization in a multiple automated guided vehicle manufacturing system via integrated heuristic algorithms. *Robotics and Autonomous Systems* 72 (2015), 131–138.

[31] Paulo R. Ferreira, Felipe S. Boffo, and Ana L.C. Bazzan. 2007. Using Swarm-GAP for distributed task allocation in complex scenarios. In *International Conference on Autonomous Agents and Multiagent Systems*. Springer, 107–121.

[32] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. 2018. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research* 61 (2018), 623–698.

[33] Merrill M. Flood. 1956. The traveling-salesman problem. *Operations Research* 4, 1 (1956), 61–75.

[34] Stan Franklin and Art Graesser. 1996. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *International Workshop on Agent Theories, Architectures, and Languages*. Springer, 21–35.

[35] Andrew Garland and Richard Alterman. 2004. Autonomous agents that learn to better coordinate. *Autonomous Agents and Multi-Agent Systems* 8, 3 (2004), 267–301.

[36] Brian P. Gerkey and Maja J. Matarić. 2004. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research* 23, 9 (2004), 939–954. https://doi.org/10.1177/0278364904045564

[37] Matthew Gombolay, Reed Jensen, Jessica Stigile, Sung-Hyun Son, and Julie Shah. 2016. Apprenticeship Scheduling: Learning to Schedule from Human Experts. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*. AAAI Press, 826–833. http://dl.acm.org/citation.cfm?id=3060621.3060736

[38] V.I. Gorodetskii. 2012. Self-organization and multiagent systems: I. Models of multiagent self-organization. *Journal of Computer and Systems Sciences International* 51, 2 (2012), 256–281.

[39] Fenton Ho and Mohamed Kamel. 1998. Learning coordination strategies for cooperative multiagent systems. *Machine Learning* 33, 2 (1998), 155–177.

[40] Yujing Hu, Yang Gao, and Bo An. 2015. Learning in Multi-agent Systems with Sparse Interactions by Knowledge Transfer and Game Abstraction. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS '15)*. IFAAMAS, 753–761.

[41] Simon Hunt, Qinggang Meng, Chris Hinde, and Tingwen Huang. 2014. A consensus-based grouping algorithm for multi-agent cooperative task allocation with complex requirements. *Cognitive computation* 6, 3 (2014), 338–350.

[42] Xiao Jia and Max Q.-H Meng. 2013. A survey and analysis of task allocation algorithms in multi-robot systems. In *International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2280–2285.

[43] Yichuan Jiang. 2016. A survey of task allocation and load balancing in distributed systems. *IEEE Transactions on Parallel and Distributed Systems* 27, 2 (2016), 585–599.

[44] Luke B. Johnson, Han-Lim Choi, and Jonathan P. How. 2016. The Role of Information Assumptions in Decentralized Task Allocation: A Tutorial. *IEEE Control Systems* 36, 4 (2016), 45–58.

[45] Luke B. Johnson, Han-Lim Choi, Sameera Ponda, and Jonathan P. How. 2012. Allowing Non-Submodular Score Functions in Distributed Task Allocation. In *51st IEEE Conference on Decision and Control (CDC)*. IEEE, 4702–4708.

[46] Luke B. Johnson, Sameera Ponda, Han-Lim Choi, and Jonathan P. How. 2010. Improving the Efficiency of a Decentralized Tasking Algorithm for UAV Teams with Asynchronous Communications. In *AIAA Guidance, Navigation, and Control Conference*. 8421.

[47] Nicolas Jozefowiez, Frédéric Semet, and El-Ghazali Talbi. 2008. Multi-Objective Vehicle Routing Problems. *European Journal of Operational Research* 189, 2 (2008), 293–309. https://doi.org/10.1016/j.ejor.2007.05.055

[48] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. 2015. Multi-robot Task Allocation: A Review of the State-of-the-Art. *Cooperative Robots and Sensor Networks 2015* 604 (2015), 31–51. https://doi.org/10.1007/978-3-319-18299-5_2

[49] Alexander Kleiner, Alessandro Farinelli, Sarvapali D. Ramchurn, Bing Shi, Fabio Maffioletti, and Riccardo Reffato. 2013. Rmasbench: benchmarking dynamic multi-agent coordination in urban search and rescue. In *International conference on Autonomous agents and multi-agent systems*. IFAAMAS, 1195–1196.

[50] Antoon W.J. Kolen, A.H.G. Rinnooy Kan, and Harry W.J.M. Trienekens. 1987. Vehicle routing with time windows. *Operations Research* 35, 2 (1987), 266–273.

[51] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. 2013. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research* 32, 12 (oct 2013), 1495–1512. https://doi.org/10.1177/0278364913496484

[52] C. Ronald Kube and Eric Bonabeau. 2000. Cooperative transport by ants and robots. *Robotics and autonomous systems* 30, 1-2 (2000), 85–101.

[53] Philippe Lacomme, Aziz Moukrim, and Nikolay Tchernev. 2005. Simultaneous job input sequencing and vehicle dispatching in a single-vehicle automated guided vehicle system: a heuristic branch-and-bound approach coupled with a discrete events simulation model. *International Journal of Production Research* 43, 9 (2005), 1911–1942.

[54] Michail G. Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton J. Kleywegt, Sven Koenig, Craig A. Tovey, Adam Meyerson, and Sonal Jain. 2005. Auction-Based Multi-Robot Routing. *Robotics: Science and Systems* 5 (2005), 98.

[55] Gilbert Laporte. 1992. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research* 59, 3 (1992), 345–358.

[56] Martin Lauer and Martin Riedmiller. 2000. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Seventeenth International Conference on Machine Learning*. Citeseer.

[57] Shen Lin and Brian W. Kernighan. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations research* 21, 2 (1973), 498–516.

[58] John D.C. Little, Katta G. Murty, Dura W. Sweeney, and Caroline Karel. 1963. An algorithm for the traveling salesman problem. *Operations research* 11, 6 (1963), 972–989.

[59] Michael L. Littman. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning (ICML)*. 157–163.

[60] Lantao Liu, Nathan Michael, and Dylan A. Shell. 2014. Fully Decentralized Task Swaps with Optimized Local Searching. *Robotics: Science and Systems* (2014).

[61] Lantao Liu, Nathan Michael, and Dylan A. Shell. 2015. Communication constrained task allocation with optimized local task swaps. *Autonomous Robots* 39, 3 (2015), 429–444. https://doi.org/10.1007/s10514-015-9481-9

[62] Lantao Liu and Dylan A. Shell. 2013. An anytime assignment algorithm: From local task swapping to global optimality. *Autonomous Robots* 35, 4 (2013), 271–286. https://doi.org/10.1007/s10514-013-9351-2

[63] Zhixing Luo, Hu Qin, and Andrew Lim. 2014. Branch-and-price-and-cut for the multiple traveling repairman problem with distance constraints. *European Journal of Operational Research* 234, 1 (2014), 49–60.

[64] Rajiv T. Maheswaran, Jonathan P. Pearce, and Milind Tambe. 2004. Distributed Algorithms for DCOP: A Graphical-Game-Based Approach.. In *Parallel and Distributed Computing Systems (PDCS)*. 432–439.

[65] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. 2004. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems*. IEEE Computer Society, 310–317.

[66] Andrei Marinescu, Ivana Dusparic, and Siobhán Clarke. 2017. Prediction-Based Multi-Agent Reinforcement Learning in Inherently Non-Stationary Environments. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 12, 2 (2017), 9:1–9:23.

[67] Robert McGill, John W. Tukey, and Wayne A. Larsen. 1978. Variations of box plots. *The American Statistician* 32, 1 (1978), 12–16.

[68] Hakim Mitiche, Dalila Boughaci, and Maria Gini. 2015. Efficient Heuristics for a Time-Extended Multi-Robot Task Allocation Problem. In *First International Conference on New Technologies of Information and Communication (NTIC)*. IEEE, 1–6.

[69] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. 2005. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161, 1-2 (2005), 149–180.

[70] Alejandro R. Mosteo and Luis Montano. 2010. A survey of multi-robot task allocation. *Instituto de Investigación en Ingenieria de Aragón, Technical Report* (2010).

[71] Ranjit Nair, Milind Tambe, Makoto Yokoo, David Pynadath, and Stacy Marsella. 2003. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Vol. 3. 705–711.

[72] Maitreyi Nanjanath and Maria Gini. 2010. Repeated auctions for robust task execution by a robot team. *Robotics and Autonomous Systems* 58, 7 (2010), 900–909.

[73] Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. 2017. A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems* 90 (2017), 55–70.

[74] Alexandru Iulian Orhean, Florin Pop, and Ioan Raicu. 2017. New scheduling approach using reinforcement learning for heterogeneous distributed systems. *J. Parallel and Distrib. Comput.* (2017). https://doi.org/10.1016/j.jpdc.2017.05.001

[75] Nunzia Palmieri, Xin-She Yang, Floriano De Rango, and Salvatore Marano. 2017. Comparison of bio-inspired algorithms applied to the coordination of mobile robots considering the energy consumption. *Neural Computing and Applications* (2017), 1–24.

[76] Liviu Panait and Sean Luke. 2005. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems* 11, 3 (2005), 387–434.

[77] Luis Paquete and Thomas Stützle. 2003. A two-phase local search for the biobjective traveling salesman problem. In *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 479–493.

[78] James Parker, Alessandro Farinelli, and Maria Gini. 2016. Max-sum for allocation of changing cost tasks. In *International Conference on Intelligent Autonomous Systems*. Springer, 629–642.

[79] Sameera Ponda, Josh Redding, Han-Lim Choi, Jonathan P. How, Matt Vavrina, and John Vian. 2010. Decentralized planning for complex missions with dynamic communication constraints. In *American Control Conference (ACC)*. IEEE, 3998–4003.

[80] Sameera S. Ponda, Luke B. Johnson, Andrew N. Kopeikin, Han-Lim Choi, and Jonathan P. How. 2012. Distributed planning strategies to ensure network connectivity for dynamic heterogeneous teams. *IEEE Journal on Selected Areas in Communications* 30, 5 (2012), 861–869.

[81] Marc Pujol-Gonzalez, Jesus Cerquides, Alessandro Farinelli, Pedro Meseguer, and Juan Antonio Rodriguez-Aguilar. 2015. Efficient inter-team task allocation in RoboCup Rescue. In *International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, 413–421.

[82] Sarvapali D. Ramchurn, Alessandro Farinelli, Kathryn S. Macarthur, and Nicholas R. Jennings. 2010. Decentralized coordination in robocup rescue. *Comput. J.* 53, 9 (2010), 1447–1461.

[83] T. Sandholm. 1998. Contract types for satisficing task allocation. In *Proceedings of the AAAI spring symposium: Satisficing models*. 23–25.

[84] Martin Savelsbergh. 1997. A branch-and-price algorithm for the generalized assignment problem. *Operations research* 45, 6 (1997), 831–841.

[85] Darren Smith, Jodie Wetherall, Stephen Woodhead, and Andrew Adekunle. 2014. A cluster-based approach to consensus based distributed task allocation. In *International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 428–431.

[86] Matthijs T.J. Spaan and Frans A. Oliehoek. 2008. The MultiAgent Decision Process toolbox: software for decision-theoretic planning in multiagent systems. In *AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM)*. 107–121.

[87] M. Statheropoulos, A. Agapiou, G. C. Pallis, K. Mikedi, S. Karma, J. Vamvakari, M. Dandoulaki, F. Andritsos, and C. L. Paul Thomas. 2015. Factors that affect rescue time in urban search and rescue (USAR) operations. *Natural Hazards* 75, 1 (2015), 57–69. https://doi.org/10.1007/s11069-014-1304-3

[88] Peter Stone and Manuela Veloso. 2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots* 8, 3 (2000), 345–383.

[89] Cynthia Sung, Nora Ayanian, and Daniela Rus. 2013. Improving the performance of multi-robot systems by task switching. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2999–3006.

[90] Katia P. Sycara. 1998. Multiagent systems. *AI magazine* 19, 2 (1998), 79.

[91] Milind Tambe. 1997. Towards flexible teamwork. *Journal of artificial intelligence research* 7 (1997), 83–124.

[92] Ming Tan. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Tenth international conference on machine learning*. 330–337.

[93] Avraam Th. Tolmidis and Loukas Petrou. 2013. Multi-objective optimization for dynamic task allocation in a multi-robot system. *Engineering Applications of Artificial Intelligence* 26, 5-6 (may 2013), 1458–1468. https://doi.org/10.1016/j.engappai.2013.03.001

[94] Joanna Turner, Qinggang Meng, and Gerald Schaefer. 2015. Increasing allocated tasks with a time minimization algorithm for a search and rescue scenario. *International Conference on Robotics and Automation (ICRA)* (2015), 3401–3407.

[95] Joanna Turner, Qinggang Meng, Gerald Schaefer, and Andrea Soltoggio. 2018. Distributed Strategy Adaptation with a Prediction Function in Multi-Agent Task Allocation. In *17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. IFAAMAS.

[96] Joanna Turner, Qinggang Meng, Gerald Schaefer, and Andrea Soltoggio. 2018. Fast Consensus for Fully Distributed Multi-Agent Task Allocation. In *The 33rd Symposium On Applied Computing (SAC '18)*. ACM/SIGAPP. https://doi.org/10.1145/3167132.3167224

[97] Joanna Turner, Qinggang Meng, Gerald Schaefer, Amanda Whitbrook, and Andrea Soltoggio. 2018. Distributed task rescheduling with time constraints for the optimization of total task allocations in a multirobot system. *IEEE transactions on cybernetics* 48, 9 (2018), 2583–2597.

[98] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. 2011. The orienteering problem: A survey. *European Journal of Operational Research* 209, 1 (2011), 1–10.

[99] Lihui Wang, Shadi Keshavarzmanesh, Hsi-Yung Feng, and Ralph O Buchal. 2009. Assembly process planning and its future in collaborative manufacturing: a review. *The International Journal of Advanced Manufacturing Technology* 41, 1-2 (2009), 132.

[100] Gerhard Weiß. 1995. Adaptation and learning in multi-agent systems: Some remarks and a bibliography. In *International Joint Conference on Artificial Intelligence*. Springer, 1–21.

[101] Gerhard Weiss. 1999. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press.

[102] Amanda Whitbrook, Qinggang Meng, and Paul W.H. Chung. 2015. A novel distributed scheduling algorithm for time-critical multi-agent systems. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 6451–6458. https://doi.org/10.1109/IROS.2015.7354299

[103] Amanda Whitbrook, Qinggang Meng, and Paul W.H. Chung. 2017. Reliable, distributed scheduling and rescheduling for time-critical, multiagent systems. *IEEE Transactions on Automation Science and Engineering* (2017).

[104] Michael Wooldridge and Nicholas R. Jennings. 1995. Intelligent agents: Theory and practice. *The knowledge engineering review* 10, 2 (1995), 115–152.

[105] Dayong Ye, Minjie Zhang, and Athanasios V. Vasilakos. 2017. A survey of self-organization mechanisms in multiagent systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47, 3 (2017), 441–461.

[106] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on knowledge and data engineering* 10, 5 (1998), 673–685.

[107] Chongjie Zhang and Victor R. Lesser. 2011. Coordinated Multi-Agent Reinforcement Learning in Networked Distributed POMDPs. In *Twenty-Fifth Conference on Artificial Intelligence (AAAI-11)*.

[108] Wanqing Zhao, Qinggang Meng, and Paul W.H. Chung. 2015. A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario. *IEEE transactions on cybernetics* 46, 4 (2015), 902–915.

[109] Xiaoming Zheng and Sven Koenig. 2009. K-Swaps : Cooperative Negotiation for Solving Task-Allocation Problems. In *International Joint Conference on Artificial Intelligence*, Vol. 9. 373–379.