University of Southampton

Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

Large-scale, Dynamic and Distributed Multi-agent Coordination for Real-time Systems

by

Luca Capezzuto

A thesis for the degree of
Doctor of Philosophy

November 2021

University of Southampton

Abstract

Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

Doctor of Philosophy

Large-scale, Dynamic and Distributed Multi-agent Coordination for Real-time Systems

by Luca Capezzuto

The *Coalition Formation with Spatial and Temporal constraints Problem* (CFSTP) is designed to characterise scenarios at the intersection between task allocation and coalition formation. In this model, tens of heterogeneous agents are deployed over kilometre-wide areas to carry out thousands of tasks, each with its deadline and workload. To maximise the number of tasks completed, the agents need to cooperate by forming, disbanding and reforming coalitions.

In this thesis, we start with an in-depth analysis of *Coalition Formation with Look-Ahead* (CFLA), the state-of-the-art CFSTP algorithm. We outline its main limitations, based on which we derive an extension called CFLA2. We show that we cannot eliminate the limitations of CFLA in CFLA2, hence we also develop a novel algorithm called *Cluster-based Task Scheduling* (CTS), which is the first to be simultaneously anytime, efficient and with convergence guarantee. We empirically demonstrate the superiority of CTS over CFLA and CFLA2, and propose S-CTS, a simplified but parallel and more efficient variant. In problems generated by the RoboCup Rescue Simulation, S-CTS can compete with the high-performance Binary Max-Sum and DSA algorithms, while being up to two orders of magnitude faster.

We then propose a minimal mathematical program of the CFSTP, and reduce it to a Dynamic and Distributed Constraint Optimisation Problem, based on which we design D-CTS, a distributed version of CTS. We create a test framework that simulates the mobilisation of firefighters, which we use to show the effectiveness of D-CTS in large-scale and dynamic environments.

Finally, to characterise scenarios in which the faster the tasks are solved, the greater the benefits, we propose the *Multi-Agent Routing and Scheduling through Coalition Formation problem* (MARSC), a generalisation of both the CFSTP and the important Team Orienteering Problem with Time Windows. We formulate a binary integer program and propose *Anytime, exact and parallel Node Traversal* (ANT), the first algorithm of its kind for both the MARSC and the CFSTP. Moreover, we define an approximate variant called ANT-ε. Both algorithms are validated in our realistic test framework, using as baselines an extended version of CTS, and an implementation of the Earliest Deadline First technique, which is typically used in real-time systems.

Contents

List of Figures

List of Algorithms

Nomenclature

Acronyms

ANT Anytime, exact and parallel Node Traversal

BFS Breadth-First Search
BinaryMS Binary Max-Sum
BIP Binary Integer Program

CF Coalition Formation
CFLA CF with Look-Ahead
CFLA2 CF with improved Look-Ahead
CFSTP CF with Spatial and Temporal constraints Problem
COP Constraint Optimisation Problem
CSG Coalition Structure Generation
CTS Cluster-based Task Scheduling

D-CTS Distributed CTS
DCOP Distributed COP
Dec-MDP Decentralised MDP
DFS Depth-First Search
DSA Distributed Search Algorithm
DynDCOP Dynamic DCOP

EDF Earliest Deadline First

FMS Fast Max-Sum

LFB London Fire Brigade

MARSC Multi-Agent Routing and Scheduling through CF problem
MAS Multi-Agent System
MDP Markov Decision Process
MIP Mixed Integer Program
MRS Multi-Robot System
MRTA Multi-Robot Task Allocation

NDCS Normally Distributed Coalition Structures

POMDP Partially Observable MDP

RCRS RoboCup Rescue Simulation

S-CTS Simplified CTS

TOP Team Orienteering Problem
TOPTW TOP with Time Windows
TSP Travelling Salesman Problem

UAV Unmanned Aerial Vehicle
USV Unmanned Surface Vehicle

VRP Vehicle Routing Problem
VRPP VRP with Profits

Symbols

a agent
A agent set
A^t_{free} agents that are free at t
A^t_v agents that at t can reach v by γ_v
α_v earliest time of v
β_v soft latest time of v

C coalition
Γ set of coalition allocations
γ_v hard latest time of v

D task demands
$\mathcal{D}$ DynDCOP
$\mathcal{D}_t$ DCOP at t
δ_v task degree (used by CFLA and CFLA2)

f_j^t j-th DCOP function at t
ϕ_v benefit of v

L all possible task and agent locations
l_a^t location of a at t
L_v possible locations of v

$\mathcal{N}$ Normal distribution

P power set
P*rec* Set of task precedences
$\psi_{v,t}$ penalty of performing v during t

ρ agent travel function

t time
t_{max} maximum problem time

u coalition value function
$\mathcal{U}$ Uniform distribution

v task
V task set

w_v workload of v

$\boldsymbol{x}$ BIP solution
$\boldsymbol{x}_i$ BIP singleton solution
χ_i^t i-th DCOP variable at t
$x_{v,t,\mathrm{C}}$ CFSTP indicator variable
$x_{v,l,t,\mathrm{C}}$ MARSC indicator variable

y_v CFSTP task indicator variable
$y_{v,l}$ MARSC task indicator variable

Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research. I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published in a number of journal and conference papers (see Section 1.4 for a detailed list).

Signed:

Date:

Acknowledgements

'The best theory is inspired by practice. The best practice is inspired by theory.'
— Donald E. Knuth, *Selected Papers on Computer Science*, 1996

'We have a habit in writing articles published in scientific journals to make the work as finished as possible, to cover up all the tracks, to not worry about the blind alleys or describe how you had the wrong idea first, and so on. So there isn't any place to publish, in a dignified manner, what you actually did in order to get to do the work.'
— Richard P. Feynman, *Nobel Lecture*, 1966

First of all, I would like to thank my advisors, Gopal Ramchurn and Danesh Tarapore, who allowed me to embark on this path, and helped me to grow both as a researcher and as a professional. I am grateful to Klaus-Peter Zauner, Tim Norman and Jie Zhang for examining my progress and giving me valuable advice. Thanks also to Alessandro Farinelli and Mohammad Divband Soorati for the edifying discussions, and to Ryan Beal and Shaun Lamb for the good times we had together in Chania and in the best breweries of Southampton.

This research is supported by the EPSRC grant EP/R2115315/1 and the AXA Research Fund. I would also like to acknowledge the use of the Iridis High Performance Computing Facility, and associated support services at the University of Southampton.

To all the people with whom I have made friends, or even shared just a laugh, in student societies such as Italian Society, MexSoc, CathSoc, Live Music Society, SLAPS and Salsa Society, to name a few. In particular, I would like to mention my carefree Sundays of roast, Mexican chocolate and Neapolitan memes with Pierfrancesco and Davide, which helped me to feel less homesick and to cope with pressing commitments.

To the López de la Cruz family, who welcomed me warmly and allowed me to work in peace during uncertain months. To Elizabeth, with whom I went hiking in mountains such as Cerro de la Silla, Cerro Agujerado and Cerro de Chipinque, and then ended the day by going for chilaquiles. To Ixora and all my other Mexican friends, with whom I spent many fun evenings.

To my family, in particular my mother and my little sister, true forces of nature, who have always helped and supported me.

To my beautiful love Ilse, without whom I would not be here. *Te cielo.*

Dedicated to the women of my life, in order of appearance: mum Agnese, granny Luigia, aunt Nunzia, little sister Annaclara, and amorcita Ilse.

Chapter 1

Introduction

A main class of multi-agent assignment problems is *cooperative coordination*, which considers the synergies between agents that have common goals [Shoham and Leyton-Brown 2008]. In multi-robot systems, it is called *Multi-Robot Task Allocation* (MRTA) [Gerkey and Matarić 2004]. Within this class, we are interested in problems involving both routing and scheduling, which are crucial in areas of increasing importance such as precision agriculture, environmental monitoring, warehouse automation, pickup and delivery, last-mile planning, space exploration, and disaster response [Nunes, Manner et al. 2017].

In this thesis, we study large-scale, dynamic and distributed cooperative coordination, with focus on disaster response scenarios [Kitano and Tadokoro 2001; Murphy 2014]. We explain the challenges of disaster response in Section 1.1, and state our problem in Section 1.2. We summarise our research objectives and methodology in Section 1.3. Finally, we highlight our contributions in Section 1.4, and outline the structure of the document in Section 1.5.

1.1 Motivating Domain: Disaster Response

The information age is plagued by all kinds of disasters. Through the screens of our smart devices, we helplessly witness events such as wildfires, explosions, earthquakes, and volcanic eruptions. Although there is no generally accepted terminology, modern disaster management can be divided into the following phases [Alexander 2002; Hawe et al. 2012]:

1. *Mitigation*: reducing or eliminating the possibility of a disaster;
2. *Preparation*: equipping humans with the means to increase their survival and to minimise the losses during a disaster;
3. *Response*: reducing or eliminating the impact of a disaster;
4. *Recovery*: restoring the situation to the state prior to the impact of a disaster.

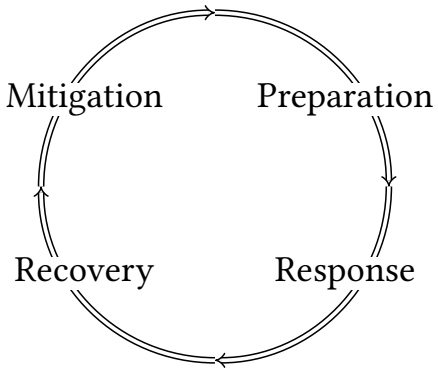


Figure 1.1 The disaster management cycle [Alexander 2002]. The mitigation and preparation phases try to prevent disasters with measures such as keeping evacuation plans up to date and minimising safety issues. The response phase focuses mainly on saving and safeguarding human lives, while the recovery phase is dedicated to repair damage, which in some cases can take years.

Regardless of how effective the mitigation and preparation phases may be, they are unable to eliminate every possible hazard [Coppola 2006]. Thus, the response phase plays an important role in the disaster management cycle. It consists of a set of complex actions like search and rescue, first aid and infrastructure restoration, which are carried out during periods of high stress, in severely time-constrained environments, and with limited information. In this phase, it is fundamental to plan firmly and act as fast as possible, since any delay can lead to further tragedy and destruction.

Consider the aftermath of a disaster, either natural, like the Cumbre Vieja volcanic eruption in 2021, or man-made, like the Beirut explosion in 2020. Different actors can be present, such as first responders, aid organisations, news reporters and local citizens. The first objective is to assess the situation and determine what are the problems to solve. For this purpose, UAVs may be deployed to patrol the affected areas, sensors may be used to monitor the environment, and citizens may contribute using social media platforms such as Ushahidi [Okolloh 2009] or Google Crisis Response. From heterogeneous sources, a great variety of content will be generated, such as maps, forecasts and reports. Once a set of tasks has been identified, a set of agents will be defined to perform them, each with its specific resources and capabilities. Then, the response phase will take place in the following steps [Jennings et al. 2014]:

1. *Formation*: agents jointly define how tasks are allocated. To encourage commitment, incentive schemes could be put in place (e.g., completing high-priority tasks could be paid double);

2. *Operation*: agents perform the tasks allocated to them. The execution is not straightforward: more critical tasks may appear, new agents or resources may become available, or current agents may disengage (e.g., a UAV could fall due to a malfunction and be destroyed). Therefore, agents must monitor the system status continuously, and redefine assignments if necessary;

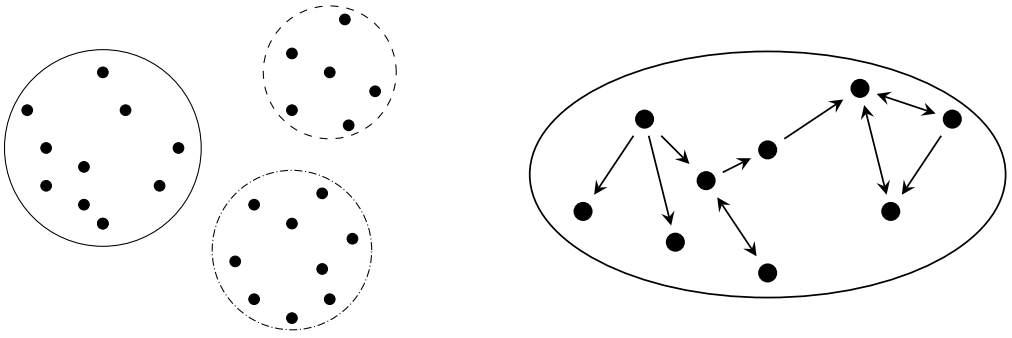
Figure 1.2 Examples of cooperative coordination in disaster response. From left to right: USV inspecting piers after Hurricane Wilma [Murphy 2014]; the Colossus robot assisting the Paris Fire Brigade in the fight against the 2019 Notre Dame Cathedral fire [IEEE Robots 2019]; fire-fighting UAVs in the Dazu District, China [iChongqing 2020].

3. *Disbandment*: the collective is disbanded either when all tasks have been successfully completed, or the emergence of some condition prevents continuation (e.g., a sudden fire spread impedes access to an area of interest, and no firefighters are available at the moment). After that, any rewards, such as payments, transfer of privileges or additional information, are distributed among agents.

As can be seen, disaster response provides a comprehensive example of the various challenges of cooperative coordination. In fact, it is a central class of case studies for the multi-agent/robot systems community [Dadvar and Habibian 2021; Kitano and Tadokoro 2001; Murphy 2014, 2016; Murphy, Tadokoro and Kleiner 2016], as well as for AI in general [Imran et al. 2014]. Based on this domain, we identify our problem below.

1.2 Problem Statement

We focus on the *Coalition Formation with Spatial and Temporal constraints Problem* (CFSTP) [Ramchurn, Polukarov et al. 2010]. We use the definitions of *coalition* and *coalition formation* given by [Horling and V. Lesser 2005]. Hence, a coalition is a flat and task-oriented organisation of agents, short-lived and disbanded when no longer needed, while coalition formation is a consequence of the emergent behaviour of the system [Mataric 1993]. Regardless of its current location, an agent is considered part of a coalition from the moment it is assigned a task, to the moment the task is completed. We consider coalitions to be different from teams (Figure 1.3). Moreover, based on [Nunes, Manner et al. 2017, Section 4.5], we consider a system to be *real-time* if it may have *hard* deadlines, which cannot be violated, and possibly also *soft* deadlines, which can be violated with some penalty. In the CFSTP, tasks (e.g., save victims or put out fires) have to be assigned to agents (e.g., ambulances or fire brigades). The assignment is determined by the spatial distribution of tasks in the disaster area, the time needed to reach them, the workload they require (e.g., how large a fire is) and their deadlines (e.g., estimated time left before victims perish). In addition to these constraints, the number of agents may be much smaller than the number of tasks, hence they need to cooperate with each other by forming, disbanding and reforming coalitions over time [Epstein and Bazzan 2011; Shehory



(a) Coalitions. **(b)** A team.

Figure 1.3 Differences between coalitions and teams [Horling and V. Lesser 2005]. Once formed, a coalition becomes an atomic entity that exists for a limited time, such as political parties joining together to form a government. In contrast, a team is a long-term organisation in which agents may have different roles, as in the game of football.

and Kraus 1998]. The objective of the CFSTP is to define which tasks (e.g., sites with the most victims and the strongest fires) to allocate to which coalitions (e.g., the fastest ambulances and fire engines with the largest water tanks), in order to maximise the number of tasks completed.

Despite having similarities with classic problems such as Generalised Assignment Problem [Ross and Soland 1975] and Job-Shop Problem [Brucker 2007], the importance of the CFSTP lies in the fact that it was the first generalisation of the *Team Orienteering Problem* (TOP) to consider coalition formation [Ramchurn, Polukarov et al. 2010, Section 4.2]. For this reason, it has been applied in contexts such as human-agent collectives [Ramchurn, F. Wu et al. 2016], multi-UAV exploration [Baker et al. 2016], and law enforcement [Nelke, Okamoto and Zivan 2020; Tkach and Amador 2021].

Within this problem class, our interest is in algorithms that are anytime (i.e., which can return partial solutions if they are interrupted before completion), efficient, and have theoretical properties. In other words, approximate algorithms [Papadimitriou 1993]. The reason is that they are fundamental in real-time domains, where it is necessary to have provable guarantees, but it may be computationally not feasible or economically undesirable to produce an optimal solution [Calvaresi et al. 2021; Zilberstein 1996]. In particular, as we said above, the faster the disaster response, the lower the losses incurred. We also assume that the agents are situated in a distributed system [Van Steen and Tanenbaum 2017] with a *dynamic evolution of the environment*[1] [Fioretto, Pontelli and Yeoh 2018], that is:

1. It must be possible for agents to reach a solution through local interaction, rather than relying on a centralised solver [Ramchurn, Farinelli et al. 2010];

2. At any time, agents can join in or leave, and new tasks can appear.

If the problem is solved in a centralised fashion, then the designated solver has to be informed whenever there is a change, recompute the solution, and redistribute it to all agents. In real-time

[1] Also referred to as *open* systems [Hewitt 1990]. For brevity, we call them *dynamic environments*.

domains such as disaster response, this leads to three major issues. First, a centralised solver is a single point of failure that makes the system fragile and not robust to unexpected events, such as malfunctions or communication disturbances between agents far apart [Petcu 2007]. Second, if the agents have limited computational resources and the problem is not small, electing a centralised solver might not be possible, while distributing computations always improves scalability [Van Steen and Tanenbaum 2017]. Third, a centralised approach might not be as effective as a distributed one, given that the situation can evolve rapidly and there could be significant communication delays [Mailler, Zheng and Ridgway 2018].

1.3 Research Objectives and Methodology

We aim to develop models, algorithms and test frameworks for multi-agent coordination, with the following requirements.

R1 *Decentralised autonomy*: the agents do not have to rely on a single point of failure. This is particularly important for disaster scenarios, where unreliable infrastructures and scarce supplies imply that lost resources cannot be replaced.

R2 *Cooperation*: the agents have to coordinate their actions so to maximise their performance. Cooperation is also necessary when tasks require combined skills. For example, to extract survivors from the rubble of a collapsed building, rescue robots detect life signs with their sensors, firefighters dig and paramedics load the injured into ambulances. Therefore, the solutions must focus on achieving a global objective.

R3 *Scalability*: the algorithms have to target scenarios with tens to thousands of agents and tasks, in order to be usable in future contexts where mass deployment of agents (e.g., robot swarms) will be common.

R4 *Resilience*: agents and resources could be added and removed at any time, either by a human supervisor or due to unexpected failures. The algorithms must be able to produce feasible solutions even in such situations. If the solution quality degrades, it must happen in a controlled way.

Moreover, motivated by the disaster response domain, and intending to address the main open issues in the MRTA literature [Nunes, Manner et al. 2017, Section 9.2], we make the following assumptions.

A1 Tasks have spatio-temporal constraints (e.g., the buildings on fire are scattered throughout the city, and each will burn out completely after a certain amount of time). In particular, each task has a time window, with *both* a soft and a hard deadline.

A2 There may be task precedences, such as when firefighters have to clear a road to allow ambulances to enter an area, and heterogeneous task weights, to capture situations in which some tasks may be more critical or urgent than others (e.g., saving lives has a greater benefit than clearing the rubble).

A3 Tasks may have multiple possible locations. This allows to characterise situations in which, for example, ambulances can transport the injured to different hospitals, or survivors can be transferred to various evacuation centres.

A4 Tasks require specific resources to be performed (e.g., it is estimated that the fire x requires at least y quantity of coolant to be extinguished).

A5 The environment is dynamic. For instance, at any moment, new fires could break out, or other buildings could collapse, hence first responders must be ready to deploy to other areas.

According to [Fioretto, Pontelli and Yeoh 2018], to date the main disciplines that have been used for solving cooperative coordination problems are game theory [Myerson 1991; Osborne and Rubinstein 1994], decision theory [Keeney and Raiffa 1993] and constraint programming [Dechter 2003; Rossi, Van Beek and Walsh 2006].

Usually, cooperative game theory models do not have a distributed design[2], or are based on the impractical assumption that calculations have no cost. On the other hand, decision theory is afflicted by super-polynomial complexity [Bellman 2003], thus designing scalable algorithms (Requirement R3) imposes additional assumptions that may be too limiting for realistic scenarios [Jesus Cerquides et al. 2013; Diederich 2001].

Hence, we choose to work with constraint programming, which, as we will show in the following chapters, allows us to meet all our research objectives. In Chapter 2, we review each of the above approaches and give a thorough motivation of our choice.

1.4 Research Contributions

We start with an in-depth analysis of *Coalition Formation with Look-Ahead* (CFLA), the state-of-the-art CFSTP algorithm. We outline two main limitations. First, its time complexity is exponential in the number of agents. Second, as we show, its look-ahead technique is not effective in real-world scenarios, such as dynamic environments, where new tasks can appear at any time. To achieve better performance, we define an extension called CFLA2. However, since we cannot eliminate the design limitations of CFLA in CFLA2, we also develop a novel algorithm called *Cluster-based Task Scheduling* (CTS), the first to be simultaneously anytime,

[2]Rather, there are models that combine non-cooperative game theory with distributed constraint optimisation [Chapman, Micillo et al. 2010; Chapman, Rogers and Jennings 2011; Chapman, Rogers, Jennings and Leslie 2011; Zou and Xi 2021].

efficient and with convergence guarantee. We empirically show that, in settings where the look-ahead technique is highly effective, CTS completes up to 30% (resp. 10%) more tasks than CFLA (resp. CFLA2), while being up to four orders of magnitude faster. We also propose S-CTS, a simplified but parallel and more efficient variant of CTS. In problems generated by the RoboCup Rescue Simulation [Kitano and Tadokoro 2001], S-CTS is at most 10% less performing than high-performance algorithms such as Binary Max-Sum [Pujol-Gonzalez, Jesus Cerquides, Farinelli, Meseguer and Juan A. Rodriguez-Aguilar 2014] and DSA [W. Zhang et al. 2005], but up to two orders of magnitude faster.

We then go on to identify the main issues in the CFSTP literature, namely, its original mathematical programming formulation, which is lengthy and difficult to implement, and the lack of a distributed, dynamic and scalable algorithm. Consequently, we propose a minimal mathematical program, and reduce the CFSTP to a *Dynamic and Distributed Constraint Optimisation Problem* (DynDCOP) [Fioretto, Pontelli and Yeoh 2018], on which we design D-CTS, a distributed version of CTS. Using public London Fire Brigade records [London Datastore 2021a,b], we create a large dataset and a test framework that simulates the mobilisation of firefighters in dynamic environments. In problems with up to 150 agents and 3000 tasks, compared to DSA-SDP, a state-of-the-art distributed algorithm [Zivan, Okamoto and Peled 2014], D-CTS completes approximately the same percentage of tasks, but with the advantage of being one order of magnitude more efficient in terms of communication overhead and time complexity.

Finally, to characterise scenarios in which the faster the tasks are solved, the greater the benefits, we propose the *Multi-Agent Routing and Scheduling through Coalition Formation problem* (MARSC), a generalisation of both the CFSTP and the important *Team Orienteering Problem with Time Windows* (TOPTW) [Vansteenwegen and Gunawan 2019]. We formulate a binary integer program [Wolsey 2020] and propose *Anytime, exact and parallel Node Traversal* (ANT), the first MARSC algorithm of its kind, which is also the first exact CFSTP algorithm. Moreover, we define an approximate variant called ANT-ε. On a machine with RHEL 7.9 operating system, a 2 GHz CPU with 40 threads, and 24 GB of DDR4-2666 SDRAM, ANT can find optimal solutions to problems with 2 agents and up to 40 tasks in less than 25 minutes. With the same operating system and CPU, but with 187.5 GB RAM, the industry-leading CPLEX solver version 20.1 runs out of memory and crashes. On the other hand, in problems with 150 agents and up to 3000 tasks, ANT-ε finds 2.6 times better median solutions and is 3.65 times faster than an extended version of CTS.

Since the MARSC generalises or can be reduced to important combinatorial optimisation problems (Figure 1.4), our results can also be applied to widely studied sub-problems such as the TOPTW, and be adapted to domains that are similar to or less challenging than disaster response, such as those mentioned at the beginning of this chapter (Page 1).

Our work has produced the following articles.

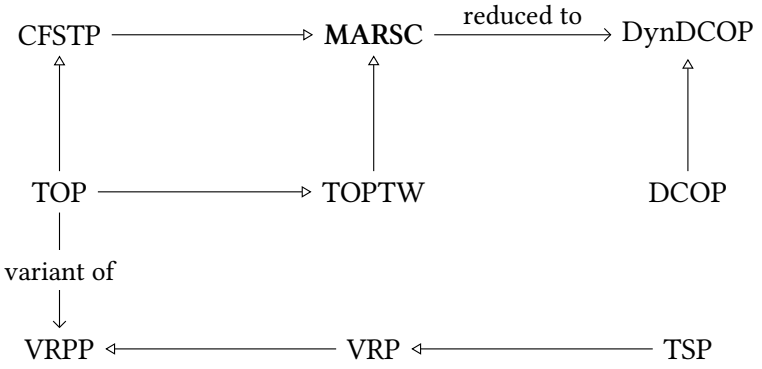


Figure 1.4 Class diagram of the relationships between the combinatorial optimisation problems that we consider. We generalise the CFSTP into the MARSC, and show how both can be reduced to a DynDCOP, an extension of the DCOP [Fioretto, Pontelli and Yeoh 2018]. We also show that the MARSC generalises the TOPTW, an extension of the TOP, the latter generalised by the CFSTP (Section 1.2). The TOP is a variant of the *Vehicle Routing Problem with Profits* (VRPP), which generalises the *Vehicle Routing Problem* (VRP) and the *Travelling Salesman Problem* (TSP) [Vansteenwegen and Gunawan 2019].

1. Capezzuto, Luca, Danesh Tarapore, and Sarvapali D. Ramchurn. *Anytime and Efficient Coalition Formation with Spatial and Temporal Constraints.* EUMAS-AT 2020: Multi-Agent Systems and Agreement Technologies, pp. 589 – 606 (2020). Springer, Cham.

2. Capezzuto, Luca, Danesh Tarapore, and Sarvapali D. Ramchurn. *Anytime and Efficient Multi-agent Coordination for Disaster Response.* SN Computer Science 2.3, pp. 1 – 15 (2021).

3. Capezzuto, Luca, Danesh Tarapore, and Sarvapali D. Ramchurn. *Multi-Agent Routing and Scheduling through Coalition Formation.* 12th Workshop on Optimisation and Learning in Multi-Agent Systems (OptLearnMAS), held at the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021).

4. Capezzuto, Luca, Danesh Tarapore, and Sarvapali D. Ramchurn. *Large-scale, Dynamic and Distributed Coalition Formation with Spatial and Temporal Constraints.* 18th European Conference on Multi-Agent Systems (EUMAS 2021), Revised Selected Papers, pp. 108 – 125 (2021). Springer, Cham.

Chapter 4 is composed of Articles 1 and 2, Chapter 5 is Article 4 with minor exposition improvements, while Chapter 6 is an extension of Article 3.

1.5 Thesis Outline

The remaining chapters are structured as follows.

Chapter 2 A survey of multi-agent coalition formation for task allocation in the research fields identified in Section 1.3. Its has two purposes. First, it motivates in detail which field we

choose to work with. Second, it shows that, although many existing models come close to our objectives, no one can satisfy them all, which motivates the proposal of the MARSC in Chapter 6.

Chapter 3 The topics on which the following chapters will be based: our constraint programming formulation of the CFSTP; the CFLA algorithm, and the original mixed integer programming formulation of the CFSTP.

Chapter 4 The CFLA2 algorithm, a novel variant of CFLA. Design of the CTS algorithm, the new state-of-the-art CFSTP algorithm. Definition of the parallel variant S-CTS, and empirical evaluation with the RoboCup Rescue Simulation against the high-performance Binary Max-Sum and DSA algorithms.

Chapter 5 A minimal binary integer program of the CFSTP, and a reduction to the DynDCOP formalism. Design of D-CTS, a distributed version of CTS, and empirical evaluation on our large-scale and realistic test framework based on London Fire Brigade records.

Chapter 6 Formulation of the MARSC, a generalisation of the CFSTP and the TOPTW that can be used in real-time domains. Design of ANT, the first anytime, exact and parallel MARSC algorithm, as well as the approximate variant ANT-ε.

Conclusions Summary of the strengths and limitations of our work, followed by a list of possible directions for future work.

Chapter 2

Literature Review

This chapter reviews the areas identified in Section 1.3, namely, game theory, decision theory and constraint programming. Based on the classifications of [Aziz et al. 2021; Cao et al. 2013; Jesus Cerquides et al. 2013; Farinelli, Iocchi and Nardi 2004; Gini 2017; Jahn et al. 2020; Khamis, Hussein and Elmogy 2015; Korsah, Stentz and Dias 2013; Miloradović et al. 2019; Nunes, Manner et al. 2017; Parker, Rus and Sukhatme 2016; Ponda et al. 2015; Skaltsis, Shin and Tsourdos 2021; Verma and Ranga 2021; Ye, M. Zhang and Vasilakos 2016], we identify 3 fundamental approaches to cooperative coordination: coalition formation in game theory (Section 2.1); Markov decision processes in decision theory[1] (Section 2.2), and distributed constraint optimisation problems in constraint programming (Section 2.3).

For each approach, we highlight the main results that are relevant to our objectives (Section 1.3). Then, in Section 2.4, we explain which approach we choose to work with in the following chapters. Finally, in Section 2.5, we list the limitations of our starting model (Section 1.2) in relation to our objectives, and show that no work proposed so far is able to address all of them. This motivates the introduction of the novel model in Chapter 6.

2.1 Coalition Formation

Task allocation through *Coalition Formation* (CF) models scenarios where tasks cannot be performed by a single agent [Shehory and Kraus 1998]. It consists of 3 phases [Sandholm, Larson et al. 1999]:

1. *Coalition Structure Generation* (CSG): partitioning of agents into exhaustive and disjoint sets such that only agents within the same set cooperate. Such sets are called *coalitions*, while the partition is called *coalition structure*;

[1] More precisely, in the subfield of multi-agent planning [Torreño et al. 2017].

2. *Coalition value calculation*: for each coalition C, definition of a measure of the expected outcome that could be derived if C was formed. Once the calculation is done, the decision about how to optimally form coalitions can be taken, which depends on the given problem;

3. *Revenue distribution*: determination of the rewards for each agent in order to get *stable* coalitions [Rahwan, Ramchurn et al. 2009]. A coalition is stable if its agents have no incentive to deviate from it. This property is also known as *incentive compatibility* [Nisan et al. 2007, Section 9.3.2].

The most important phase is that of CSG, since the number of possible coalition structures is exponential in the number of agents. Specifically, the time complexity required to find an optimal coalition structure is $O(a^a)$ and $\omega(a^{a/2})$, where a is the number of agents. Furthermore, the problem is NP-complete [Sandholm, Larson et al. 1999]. For this reason, a wide range of approximate algorithms has been developed so far, using approaches from mathematical programming, stochastic search and machine learning. In the remainder of this section, we will review some important works that are of interest to our context. We refer to [Rahwan, T. P. Michalak et al. 2015] for a thorough survey on CSG.

Among the first and best-known attempts to study algorithmic aspects of CSG, we find [Shehory and Kraus 1998]. They developed decentralised, anytime and dynamic algorithms with low ratio bounds and low computational complexity, where coalitions may have a precedence order. Their limitation is the assumption that coalition resources can be transferred between agents, which reduces the solution space.

An optimal solution to the CSG problem has been given by [Rahwan, Ramchurn et al. 2009]. They represent the solution space through integer partition [Andrews and Eriksson 2004] and solve the problem using a branch-and-bound technique. The limitation of this approach is that, in the worst case, the number of coalition structures to be searched is intractable. As a result, the algorithm is not scalable (Requirement R3).

A CF model with incomplete information, limited computational resources, and time constraints was proposed by [Kraus, shehory and Taase 2003, 2004]. Their mechanism consists of an auction protocol and 2 heuristic algorithms. Experimental results show that their algorithms are stable and increase social welfare, compared to previous strategies. There are 2 limitations in the work. First, it focuses specifically on the Request For Proposal domain, where the tasks are independent. Therefore, it cannot be used in settings where there may be task precedences (Assumption A2). Second, the algorithms have no theoretical guarantees, while we aim at approximate algorithms (Section 1.3).

A variant of CSG is the *Graph-Constrained Coalition Formation* (GCCF) [Myerson 1977], where agents have a graph of relationships G and a coalition C is considered feasible if all agents in C are connected by a subgraph of G induced by C. This formulation allows solving a wide range of real-world problems. An interesting one is the Social Ridesharing problem [Bistaffa, Farinelli and Ramchurn 2015], in which a community of commuters (e.g., riders and drivers) need to form

coalitions (e.g., joining in cars) while meeting the constraints imposed by the social network (e.g., users prefer to ride with friends). The objective is to minimise associated transportation costs, like travel time and fuel. The model has been extended to also allow temporal constraints, such as desired pick-up and arriving times [Bistaffa, Farinelli, Chalkiadakis et al. 2017]. For a survey on real-world GCCF applications, we refer to [Bistaffa, Farinelli, Jesús Cerquides et al. 2017]. Such approaches can cope with spatio-temporal and resource constraints (Assumptions A1 and A4). Nonetheless, they are offline, which implies that situations where agents can join in or leave cannot be considered (Requirement R4 and Assumption A5). [Flammini et al. 2018] have proposed an online GCCF model in which the graph is weighted and agents are assigned to coalitions irrevocably. However, their model cannot solve problems where the graph is unweighted or coalitions can be modified (Requirement R4).

Another variant of CSG is the *Overlapping Coalition Formation* (OCF) model [Chalkiadakis, Elkind, Markakis et al. 2010], where agents may be involved in more than one task and thus may distribute their resources among multiple coalitions. There is no inherent superadditivity assumption, therefore it is possible to capture scenarios in which not always any pair of coalitions is best off by merging into one[2]. To date, however, there is still no characterisation of scenarios where overlapping coalitions can naturally arise (Assumption A5). [Zick, Chalkiadakis and Elkind 2012] showed that the OCF can be reduced to the Unbounded Knapsack Problem [Martello and Toth 1990], thus proving that it is NP-complete. They also identified that the computational complexity depends on the amount of resources possessed by each agent, the maximum coalition size, and the pattern of interaction among agents. Furthermore, they proposed tractable subproblems with discrete resources and limited interactions.

[Rahwan, T.-D. Nguyen et al. 2013] proposed the *Coalition-Flow Network* (CF-NET) model, where a characteristic function game can be represented as a network flow problem. In the CF-NET model, the value of a coalition is associated with both the execution of a given task and the types of the agents involved. Agents can participate in multiple coalitions simultaneously, which allows to consider both CSG and OCF problems. In addition, each agent has a limit on the resources that can use to execute tasks, thus there is an upper bound on the number of coalitions it can join. An anytime approximate algorithm is proposed, which provides worst-case guarantees on the solution quality. Although it permits to switch between cases with non-overlapping and overlapping coalitions efficiently, with and without agent types, the work does not consider scenarios where the order in which agents join a coalition can affect the value of the coalition [T. P. Michalak et al. 2014]. Hence, it is not suitable for real-time domains such as disaster response.

A typical assumption in CF is that the values of potential coalitions are known with certainty, and agents have complete information on the capabilities of potential partners. [Chalkiadakis and Boutilier 2004, 2012] have proposed a model in which both coalition values and agent capabilities are uncertain, and Bayesian *Reinforcement Learning* (RL) [Ghavamzadeh et al. 2015]

[2]That is, scenarios in which the emergence of the *grand coalition*, or the coalition of all agents, is either not guaranteed or impossible [Sandholm, Larson et al. 1999; Sandholm and V. R. T. Lesser 1997].

is used to reduce such uncertainties. The rationale for using RL is that agents are supposed to interact between each other by repeatedly exchanging messages, therefore the more they interact, the more accurate their beliefs about each other's capabilities. Their stability criterion, called *Bayesian core*, makes agent *a* choose coalition C based not only on the value of C, but also on its *value of information*, defined as the quantity of information that *a* can learn about other agents if C was formed. Since the proposed inferential process does not consider the cost of computation, it is not clear how the model can handle large-scale problems (Requirement R3). Another limitation is that RL approaches, and machine learning in general[3], are inapplicable to our domains of interest due to their initial training phase: before they can generate feasible solutions, the problem formulation may change, thus new training may become necessary [Tsimenidis 2020]. Moreover, such approaches are notoriously sample-inefficient, that is, they usually need millions of interactions even for the simplest problems [Yang et al. 2021]. This latency is particularly undesirable in real-time scenarios, where the situation can evolve quickly (Assumption A3).

[Krausburg, Dix and Bordini 2021] introduced the *Sequential Characteristic Function Game* (SCFG), which extends the CSG problem to a total order of structures defined by binary relations. A variant of the clustering heuristic by [Farinelli, Bicego et al. 2017] is proposed to solve the problem, and 2 examples show that the model is able to characterise disaster response scenarios where agents can be organised in dynamic hierarchies of coalitions to respond to various events. Although it opens up new research directions in game theoretic CF, the SCFG is only able to meet our Assumptions A2 and A5.

[Czarnecki and Ayan Dutta 2021] use a maximum bipartite graph matching and a hedonic CSG [Bogomolnaia and Jackson 2002] to address MRTA domains that require to minimise formation costs and to maximise coalition values. Their approach is guaranteed to reach a Nash equilibrium, and capable to solve problems with thousands of robots and hundreds of tasks in seconds, which satisfies Requirement R3. However, it does not take any of our remaining objectives into account.

2.2 Markov Decision Processes

Markov Decision Processes (MDPs) [Seuken and Zilberstein 2008] are widely used in cooperative coordination [Torreño et al. 2017], given their capability to model stochastic and sequential decision making [Diederich 2001; M. Kolobov and A. Kolobov 2012; Sutton and Barto 2018], and have been successfully applied to many agent-based models, multi-agent systems and multi-robot systems. [Guestrin, Koller and Parr 2002; Kurniawati 2021; Nijs et al. 2021; R. G. Smith and Davis 1981; Stone, Sutton and Kuhlmann 2005]. Since their computational complexity

[3]Not considering open problems such as brittleness, embedded bias, catastrophic forgetting, and explainability [Choi 2021; Dehghani et al. 2021; Roy et al. 2021], as well as the non-trivial requirement to adjust parameters according to available data and target application [Vasudevan et al. 2021]. In fact, in industry it is good practice to start without machine learning [Yan 2021].

is super-polynomial in the number of agents [Bellman 2003], MDP algorithms are typically approximate [Bertsekas 2012; Busoniu et al. 2017; Geramifard et al. 2011; Mahadevan and Maggioni 2007; Sutton 1996].

In the standard MDP model, agents are able at every time step to observe the whole global state. This is not possible in presence of uncertainty, where agents can only infer information on the global state by observing the environment. The *Partially Observable MDP* (POMDP) model extends the MDP model to such domains [Seuken and Zilberstein 2008]. Solving POMDPs is much more computationally expensive than solving MDPs, because the optimal policy is a mapping from a belief space (over the state space) to the action space, rather than from the state space to the action space. Thus, the problem dimension is exponentially increased.

When the MDP and POMDP models are applied to multi-agent scenarios, the resulting models are called *Multi-agent (Partially Observable) Markov Decision Processes* (MMDP and MPOMDP, respectively). The main difference is that decisions are collective, therefore there are: a *joint* agent state space; a *joint* action space, and a *joint* observation space. It is demonstrated that the computational complexity of M(PO)MDPs is exponential in the number of agents [Redding et al. 2012]. Furthermore, a limiting assumption of these models is that each agent is required to have a *full* (MMDPs) or *partial* (MPOMDPs) *individual observability* of the global state. That is, each agent must have the same full or partial knowledge of the joint state.

When agents decide and observe locally, the assumption of individual observability is in general impractical, as it requires that every agent communicate its observations to every other agent at every time step, with no communication costs. This is typically referred to as a *free-comm* environment [Ponda et al. 2015]. However, in real-world situations like disaster response, communication might have operational constraints, such as limited bandwidth or network topology. To address this issue, various decentralised (PO)MDP variants have been proposed, but they further increase the problem dimension and thus worsen the computational complexity. The *Decentralised MDP* (Dec-MDP) model, first proposed by [Bernstein et al. 2002], replaces the assumption of full individual observability with a full joint observability requirement: the sum of all agent observations must be equal to the full observability of the global state. To do so, agents share their observations through communication. However, this overhead has NEXPTIME complexity. If the global state cannot be uniquely determined by a joint observation, the Dec-MDP becomes a Dec-POMDP [Bernstein et al. 2002]. The Dec-POMDP has been proven to be equivalent to the *Multi-agent Team Decision Problem* (MTDP) [Pynadath and Tambe 2002] in terms of computational complexity and expressiveness of representation [Seuken and Zilberstein 2008]. If a free-comm environment is assumed, then Dec-MDP and Dec-POMDP reduce to MMDP and MPOMDP, respectively [Goldman and Zilberstein 2004].

Since the communication infrastructure plays a key role in decentralised architectures, the above models have also been extended to capture agent communication protocols. The resulting variants, namely the Dec-POMDP-COM [Goldman and Zilberstein 2003] and the COM-MTDP [Pynadath and Tambe 2002], define communications as actions on which agents must make

explicit decisions. In other words, sending a message is also an action with an associated cost, and every agent has to decide which messages to send at each time step, where the decision not to send a message is modelled as a null message with zero cost. It is shown that Dec-POMDP, MTDP, Dec-POMDP-COM and COM-MTDP are all NEXPTIME-complete in the number of agents [Seuken and Zilberstein 2008].

In MDP models, the complexity of a problem is directly linked to the assumptions on observability and communication [Seuken and Zilberstein 2008]. To keep the computational complexity tractable, the problem is typically relaxed. Among the most interesting approximate MDP model, we find the *Transition Independent Dec-MDP* (TI-Dec-MDP) [Becker et al. 2004], where agents are independent and collaborate with each other through a global reward function of all agent states and actions. The TI-Dec-MDP extends a factored version of Dec-MDP, in which the global state is divided into 2 parts: information about local agent states, and information about the environmental variables of interest. In this model, the dynamics that rule the state transitions and observations of each agent are independent (i.e., they are functions only of the local agent and the world state), and agents are only coupled through the global reward function. Being based on independent agents, the TI-Dec-MDP scales in time better than other models, but it is also more space demanding, because it requires that each agent stores its complete state transition history [Becker et al. 2004; Redding et al. 2012]. Moreover, the independence assumption does not allow to explicitly model situations due to combinations of agent states (e.g., UAV collisions), even though these can be defined by assigning very large negative rewards (e.g., instead of being defined as a set of constraints, collision avoidance could be integrated into the global reward function). Other notable approximate models include:

- The decentralised sparse-interaction MDP [Melo and Veloso 2011], in which the agent state space is subdivided into 2 subsets: states where agents need to cooperate, and states where they can act independently;
- The group-aggregate Dec-MMDP [Redding et al. 2012], where each agent stores only some properties of interest regarding other agent state-action spaces (e.g., the number of agents in a particular area, or the location of a particular subset of agents);
- The auctioned POMDP [Capitan et al. 2013], in which agents solve their local version of the global problem (optimised to their preferences), then negotiate a global solution through an auction protocol;
- The POMDP with information rewards [Spaan, Veiga and Lima 2015], where agents are rewarded for reaching certain levels of belief on target state features. This model is designed for robot-assisted surveillance, where robots have to take into account the influence of actions on the environment, as well as the potential information gain;
- The POMDP with macro-actions [Amato, Konidaris et al. 2016], aimed at solving MRTA allocation problems. A *macro-action* is a set of actions necessary to complete a high-level goal, and includes low-level actions such as moving, manipulating, and perception. [A. J.

Smith et al. 2019] extended the framework using *Monte Carlo Tree Search* (MCTS) to search the action spaces and allow real-time planning;

- The mixed observed MDP [Y. Chen, Rosolia and Ames 2021], combined with a factor graph formulation and solved with the decentralised Max-Sum algorithm (Section 2.3.1).

Summing up, computational complexity in MDPs can be reduced through approximate techniques, but this yields to problem-dependent solutions. Concerning disaster response problems, important models and solution techniques are listed below:

- Multi-agent POMDP with online Bayesian learning [Allen-Williams and Jennings 2010], in which agent policies are defined through deterministic finite state machines built with Bayesian learning. Heuristics specific to search and rescue problems are used to estimate the values of future states to add;
- Multi-agent POMDP with MCTS [Amato and Oliehoek 2015], which extends the seminal online planning method by [Silver and Veness 2010] to multi-agent planning and learning domains, with particular interest in fire-fighting and sensor network problems with up to 10 agents.
- Multi-agent MDP with MCTS [Ramchurn, F. Wu et al. 2016]. The considered problem is the CFSTP [Ramchurn, Polukarov et al. 2010], extended to capture the uncertainties of a threat diffusing in an aftermath area (e.g., a radioactive cloud), and the activity of first responders. To cope with dimensionality issues, an approximate algorithm is developed using MCTS to estimate the expected value of teams. Under the same settings, [Baker et al. 2016] proposed a decentralised variant of MCTS that allows coordinating multiple UAVs in the exploration of continuous disaster spaces, with the aim of maximising the probability of discovering survivors;
- Multi-agent MDP with rejections, called k-RMMDP [Ramchurn, Huynh, F. Wu et al. 2016], which is based on the probability that a set of agents $\mathrm{T_H}$ rejects a joint action in a given state, while each other agent $\notin \mathrm{T_H}$ accepts it. A novel approach, called *Two-Pass Planning* (TPP), is proposed to solve k-RMMDPs ad hoc. Empirical evaluation shows that, in human-*in*-the-loop settings, TTP task allocations are more likely to be accepted by first responders, and also have a better completion rate with respect to human-*on*-the-loop settings;
- POMDP with online planning and active sensing [F. Wu, Ramchurn and Xiaoping Chen 2016], where Monte Carlo simulation (for one human) is combined with information-based active sensing (for multiple UAVs) using an anytime heuristic;
- Qualitative Dec-POMDP (QDec-POMDP) with iterative planning [Bazinin and Shani 2018]. A QDec-POMDP is a Dec-POMDP in which the quantitative probability distributions over state spaces are replaced with qualitative sets of states. Although this change improves the scalability, it does not alter the computational complexity;

EE / EB	Deterministic	Stochastic
Static	DCOP	Probabilistic DCOP
Dynamic	Dynamic DCOP	–

Table 2.1 DCOP models [Fioretto, Pontelli and Yeoh 2018]. The columns refer to the environment evolution (EE), while the rows refer to the environment behaviour (EB). In this thesis, we focus on the first column.

- Dec-POMDP with multi-agent reinforcement learning [H.-R. Lee and T. Lee 2021] pre-trained by behavioural cloning [H.-R. Lee and T. Lee 2021] to solve selective patient admission problems, in which an emergency department strategically reassigns some of the incoming patients to other emergency departments to preserve its medical resources for future severe patients.

None of the solutions presented above meets all of our requirements (Section 1.3), because they are either based on heuristics [Allen-Williams and Jennings 2010; Baker et al. 2016; Bazinin and Shani 2018; F. Wu, Ramchurn and Xiaoping Chen 2016] (we aim at solutions with provable guarantees), or rely on a centralised solver [Amato and Oliehoek 2015; Ramchurn, Huynh, F. Wu et al. 2016; Ramchurn, F. Wu et al. 2016] (Requirement R1 impose decentralised solutions), or are not usable in dynamic environments [H.-R. Lee and T. Lee 2021] (Assumption A5).

2.3 Distributed Constraint Optimisation Problems

A generalisation of the Distributed Constraint Satisfaction Problem [Yokoo, Durfee et al. 1998], the *Distributed Constraint Optimisation Problem* (DCOP)[4] [Farinelli, Vinyals et al. 2013] can be classified according to the following parameters [Fioretto, Pontelli and Yeoh 2018]:

- *Agent behaviour*: how agents act. It can be deterministic or stochastic;
- *Agent knowledge*: how much agents know about their state and the state of the environment. It can be total or partial;
- *Agent coordination*: how agents interact. It can be cooperative (common goals) or competitive (individual goals);
- *Environment behaviour*: how the environment reacts to the execution of an action. It can be deterministic or stochastic;
- *Environment evolution*: this parameter defines whether the problem changes over time (dynamic) or not (static).

[4] Originally abbreviated to *DisCOP* [Meisels 2007].

Table 2.1 lists the models associated with this classification. The DCOP is a direct extension to distributed environments of the *Constraint Optimisation Problem* (COP). It is characterised by static and deterministic environment, deterministic agent behaviour, total agent knowledge, and cooperative agent coordination. *Dynamic DCOPs* (DynDCOPs) are based on decision theory concepts to model dynamic environments. Given our objectives (Section 1.3), in the next sections we give a formal definition of DCOPs and DynDCOPs, along with a summary of state-of-the-art algorithms. Following [Fioretto, Pontelli and Yeoh 2018, Section 4.3], we will use the term *incomplete* to denote an algorithm that is either approximate or heuristic (i.e., not exact).

2.3.1 DCOP

Following [Petcu 2007, Section 2.1], we begin by formulating the centralised and discrete COP.

Definition 2.1 A COP is a tuple $\mathcal{P} = \langle \mathrm{X}, \mathrm{D}, \mathrm{R} \rangle$, where X is a set of n variables, $\mathrm{X} = \{x_1, \dots, x_n\}$, D is a corresponding set of finite domains, $\mathrm{D} = \{\mathrm{D}_1, \dots, \mathrm{D}_n\}$ such that $x_i \in \mathrm{D}_i$, and R is a set of t cost functions, $\mathrm{R} = \{r_1, \dots, r_t\}$, with $r_i : \mathrm{D}_{i_1} \times \cdots \times r_{i_h} \to \mathbb{R}$, $1 \leq i \leq t$, $h \leq n$. An assignment to $\mathcal{P}$ is a set of k values $d = \{d_1, \dots, d_k\}$, $k \leq n$, where $d_i \in \mathrm{D}_i$. If $k < n$, d is called *partial assignment*, and *complete assignment* otherwise. An optimal solution to $\mathcal{P}$ is a complete assignment X^* that minimises the sum of all costs:

$$\mathrm{X}^* = \arg\min_{\substack{d \in \mathrm{D} \\ |d|=n}} \sum_{r_i \in \mathrm{R}} r_i(d) \tag{2.1}$$

In the previous definition, the functions in R are soft constraints. Nonetheless, hard constraints can be imposed by defining cost functions that evaluate feasible assignments to 0, and unfeasible ones to $+\infty$. The discrete DCOP can be formulated as follows.

Definition 2.2 A DCOP is a tuple $\mathcal{P} = \langle \mathrm{A}, \mathrm{P}, \mathrm{R}^{ia} \rangle$ such that:

- A is a set of m agents, $\mathrm{A} = \{\mathrm{A}_1, \dots, \mathrm{A}_m\}$;
- $\mathrm{P} = \{\mathcal{COP}_1, \dots, \mathcal{COP}_k\}$ is a set of disjoint COPs (definition 2.1);
- Each $\mathcal{COP}_i \in \mathrm{P}$ is called the *local subproblem* of agent A_i, who owns and controls it;
- $\mathrm{R}^{ia} = \{r_1, \dots, r_t\}$ is a set of *inter-agent* cost functions defined over variables from multiple local subproblems. In particular, each $r_i \in \mathrm{R}$ expresses the value of all possible joint decisions that can be made by the agents that control the local subproblems involved in r_i. The agents involved in r_i have full knowledge of r_i and are called *responsible* for r_i.

An optimal solution to $\mathcal{P}$ is a complete assignment to all variables of all local subproblems, such that the sum of all inter-agent cost functions is minimised.

Thus, a DCOP is a multi-agent system in which each agent controls its local COP. Hard constraints can be imposed in DCOPs with the procedure described above for COPs. It is

commonly assumed that every cost function is known to all involved agents. The DCOP is NP-hard [Farinelli, Vinyals et al. 2013]. A typical data structure for representing a DCOP is the factor graph [Kschischang, Frey and Loeliger 2001; Loeliger 2004].

Definition 2.3 A *factor graph* is a bipartite graph expressing the factors of a function. The factor graph of a DCOP $\mathcal{P}$ is composed of:

- *Variable nodes*, representing the COP variables in $\mathcal{P}$;
- *Factor nodes*, representing all COP constraints and inter-agent constraints in $\mathcal{P}$;
- Undirected edges between each factor node and the variable nodes in its scope.

Other data structures to represent DCOPs are constraint graphs and pseudo-tree [Fioretto, Pontelli and Yeoh 2018]. A *constraint graph* is an undirected hypergraph[5] where nodes represent the decision variables, and edges represent the constraints. Two nodes are in the same edge if the respective variables occur in the same constraint [Rossi, Van Beek and Walsh 2006]. A *pseudo-tree* is a connected pseudo-forest, that is, an undirected connected graph that contains at most one cycle[6]. Constraint graphs allow to define priorities between variables, and pseudo-trees capture partial orders among the agents.

Depending on how decision variables are updated, DCOP algorithms are divided into synchronous and asynchronous. *Synchronous* algorithms impose an order on how agents make their decisions, typically through the data structure used for representation. In contrast, *asynchronous* algorithms allow agents to make their decisions based uniquely on their local views of the problem. Synchronous algorithms introduce dependencies between agents, but guarantee that their local views are consistent with each other. In asynchronous algorithms, agents are independent in their decision-making, thus they can process messages as they receive them, but there is no consistency guarantee on the local views. [Peri and Meisels 2013] show that inconsistent agent views may negatively impact network load and thus performance. Therefore, it is never harmful to have some degree of synchronisation.

The design of a DCOP algorithm can be of 4 types [Mahmud, Khan and Jennings 2020; Yeoh 2010]:

- *Search-based*: the search space is pruned according to problem-dependent features. This approach is typically derived from centralised solutions, such as *Breadth-First Search* (BFS) or *Depth-First Search* (DFS) [Cormen et al. 2009];
- *Inference-based*: using dynamic programming or belief propagation, each agent aggregates information from other agents to reduce its computation load;
- *Sampling-based*: the search space is sampled to generate approximate solutions through statistical inference;

[5] A generalisation of a graph in which an edge can join 2 or more vertices.
[6] Connected acyclic graphs (trees) are therefore pseudo-trees.

Algorithm	Error bound	Time complexity	Anytime	Space complexity per agent	Number of messages	Message size	Local communication
AFB	✓	$O(d^n)$	✓	$O(n)$	$O(d^n)$	$O(n)$	×
ADOPT	✓	$O(d^n)$	×	$O(n + ld)$	$O(d^n)$	$O(n)$	✓
DPOP	✓	$O(d^w)$	×	$O(d^w)$	$O(n)$	$O(d^w)$	✓
DSA	×	$O(tld)$	×	$O(l)$	$O(tnl)$	$O(1)$	✓
D-Gibbs	✓	$O(tld)$	✓	$O(l)$	$O(tnl)$	$O(1)$	✓
k-optimal	×	$O(ld^w)$	✓	$O(d^w)$	$O(n^2l)$	$O(d^w)$	×
Max-Sum	×	$O(td^l)$	×	$O(d^l)$	$O(tnl)$	$O(d)$	✓

Table 2.2 Characteristics of main DCOP algorithms [Fioretto, Pontelli and Yeoh 2018]. The top side rows refer to exact algorithms, while those on the bottom side refer to incomplete algorithms.

- *Population-based*: sets of candidate solutions are used to represent search subspaces and avoid local optima.

In addition to computational complexity, DCOP algorithms can also be characterised according to their communication overhead. For instance, the number and size of messages, size of agent neighbourhoods, and whether communication is local (i.e., the messages are sent to neighbours only) or global (i.e., the messages are sent to all). Table 2.2 shows the properties of the main algorithms that we report below. We have chosen this set on the basis of relevance to our context. For instance, we preferred ADOPT instead of OptAPO [Grinshpoun and Meisels 2008] because, although both partially centralised, OptAPO has global communication, and we excluded MGM [Maheswaran, Pearce and Tambe 2004] because it is the deterministic predecessor of DSA, which has the same computational complexity and is more used. Apart from the algorithms listed in Table 2.2, we also report 7 recent algorithms not present in the latest DCOP survey [Fioretto, Pontelli and Yeoh 2018], namely: GDBA [Okamoto, Zivan, Nahon et al. 2016], CoCoA [Leeuwen and Pawelczak 2017], ACO_DCOP [Z. Chen, T. Wu et al. 2018], COOPT [Leite and Enembreck 2019b], LSGA [Z. Chen, L. Liu et al. 2020], AED [Mahmud, Choudhury et al. 2020] and DPSA [Mahmud, Khan and Jennings 2020]. We adopt the following notation: n is the number of decision variables; d is the size of the largest decision variable domain; w is the induced width of the pseudo-tree; l is the largest number of agent neighbours; t is the number of iteration cycles (in incomplete algorithms).

AFB *Asynchronous Forward Bounding* [A. Gershman, Meisels and Zivan 2009] is an exact, asynchronous and search-based algorithm, which uses a branch-and-bound approach and a shared data structure called *Current Partial Assignment* (CPA). As the name indicates, the CPA stores the current partial assignment to the decision variables of the problem. The CPA starts empty and is extended in sequence by each agent. When an agent has to assign values to its decision variables in the CPA, it is called the *assigning agent*; agents that have not done it yet are called *unassigned agents*. Each assigning agent a_{CPA} sends a copy of the CPA *forward* to each unassigned agent. Upon receiving a CPA copy, each unassigned agent estimates a lower bound on the cost of the CPA based on its local view of the problem, and sends it to a_{CPA}. Once a_{CPA} has (asynchronously) received the estimations from all unassigned agent, it uses them to compute a new lower bound. If this new lower bound is greater than the current upper

bound, that is, the cost of the best solution found so far, a_{CPA} starts a backtracking phase. The computational complexity of AFB depends entirely on the first assigning agent a^1_{CPA}: time and message size are both $O(n)$, since a^1_{CPA} evaluates, stores and sends the value assignment of all decision variables; the number of messages is $O(n)$, as a^1_{CPA} lists all possible value combinations of the decision variables, and sends a message for each. Because assigning agents may broadcast messages, communication is not local.

ADOPT *Asynchronous Distributed OPTimisation* [Junges and Bazzan 2008; Modi et al. 2005] is an exact, asynchronous and search-based algorithm. It was the first DCOP algorithm able to provide optimal solutions through asynchronous local communication. Its operation is as follows. The agents are prioritised in a *Depth-First Search* (DFS) [Cormen et al. 2009] pseudo-tree in which each node has a single parent and multiple children. Assignments are propagated downwards the tree, while costs are propagated upwards. Additionally, threshold messages are propagated downwards to minimise redundant searches. Each agent node stores lower and upper bounds on the cost of the partial assignment, expressed by the subtree of which it is root. The DFS structure allows agents to choose partial assignments using a BFS strategy. In other words, each agent always chooses the partial assignment with the smallest lower bound. Through the flow of messages, the lower and upper bounds of each agent node are iteratively tightened until they coincide. When that happens, the node is said to be terminated. The whole procedure is completed when all nodes terminate. This is detected by the root agent, which aggregates the global cost bounds and detects the termination of its children. Similar to AFB, the time complexity of ADOPT is $O(d^n)$, since the root agent needs to evaluate all possible value combinations of all its children. Accordingly, the message size is $O(n)$. The space complexity per agent is $O(n + ld)$, where $O(n)$ is used to store the partial assignments under investigation, and $O(ld)$ is used to store the lower and upper bounds. Finally, the tree structure implies a local communication, while the BFS strategy for backtracking prevents the algorithm from being anytime.

DPOP *Distributed Pseudo-tree Optimisation Procedure* [Junges and Bazzan 2008; Petcu and Faltings 2005a] is an exact, synchronous and inference-based algorithm. Based on the Sum-Product algorithm [Kschischang, Frey and Loeliger 2001], it starts by ordering agents through a DFS pseudo-tree, then it explores the search space through a dynamic programming technique. Like ADOPT, the exploration is done via message propagation, except that the messages contain utility values, given that the algorithm is based on maximisation problems. Once the pseudo-tree is constructed, there are 2 message propagations: from leaves to root, and then from root to leaves. In the first propagation, each node aggregates the utility messages from its children, which then uses to compute its utility message to send to its parent. In the second propagation, each node computes the optimal overall utility of its variable and sends a message to its children containing its assignment. Given the pseudo-tree structure, the complexity for space, time and message size is the same: $O(d^w)$. The number of messages is $O(n)$, since each agent sends

at most n messages at each propagation phase, and communication is local. An approximate version of DPOP has been proposed by [Petcu and Faltings 2005b].

DSA The term *Distributed Stochastic Algorithm* [W. Zhang et al. 2005] identifies a class of incomplete, synchronous and search-based algorithms. The basic structure is the following. Agents initially give random utilities to their assignments, then loop through a sequence of actions until all constraints are satisfied. At loop iteration t, each agent sends its utility to the neighbours if it changed in iteration $t - 1$, then it receives their eventual new utilities. The neighbourhoods are defined by the data structure used to represent the problem. Upon receiving its neighbour messages, each agent stochastically decides whether to keep its current utility (and thus, current assignment) or change it according to some strategy, to reduce the number of violated constraints. The DSA algorithms vary in how this strategy is defined. The time complexity is O(ld), given that each agent has to calculate the utility of it assignments considering the utilities of its neighbours. The space complexity is O(l), since each agent sends a message to each neighbour. Consequently, the total number of messages is O(tnl). The size of each message is O(1), given that it only contains the value of a given assignment. Finally, agents communicate only with their neighbours. Even though DSA algorithms have no quality guarantees, their performance has been extensively demonstrated in practice, to the point that they are commonly used as a baseline in DCOP benchmarks [Fioretto, Pontelli and Yeoh 2018, Section 4.4.2]. The state-of-the-art variant is DSA-SDP [Zivan, Okamoto and Peled 2014].

D-Gibbs *Distributed Gibbs* [D. T. Nguyen et al. 2019] is an incomplete, synchronous and sampling-based algorithm that reduces the DCOP formulation to a maximum a posteriori estimation, to which it applies the Gibbs sampling process [S. Geman and D. Geman 1993] in a decentralised manner. Operating on a pseudo-tree arrangement, each agent computes a joint probability distribution of its current partial assignment, which then uses to decide its value assignments. Such assignments are propagated down to the leaves, then cost information is propagated up to the root. The process continues until convergence or a fixed number of iterations is reached. The time complexity of D-Gibbs is O(tld), as each agent computes the cost of an assignment considering the cost values received by its children. The space complexity is O(l), since each agent only needs to store the current values of its children. The total number of messages is O(tnl), given that each agent sends a message to each of its children in the first propagation phase. The size of each message is constant, because agents only send values or costs. Finally, given the pseudo-tree structure, communication is local.

k-Optimal [Pearce and Tambe 2007] is a class of incomplete, synchronous and search-based algorithms, which decomposes a DCOP into a set of subproblems, each of which involves at most k agents. The solution process continues until no subset of k or fewer agents can improve the global solution. These algorithms are anytime and guaranteed to find a lower bound on the solution quality. However, to eliminate conflicts between partial solutions, each agent may

need to communicate with every other agent. Consequently, communication is not local, and both time and space complexity are exponential in the number of agents. Such limitations are also present in the variants proposed in [Kiekintveld et al. 2010; Vinyals et al. 2011]. The k-optimality concept can also be used to characterise the solution quality of a DCOP algorithm in an *offline* manner, that is, without solving specific problem instances, and consequently providing general results [Farinelli, Vinyals et al. 2013].

Max-Sum [Farinelli, Rogers et al. 2008] is an incomplete, synchronous and inference-based algorithm. Based on a factor graph representation of the problem, it optimises the marginal costs of each decision variable through belief propagation, similar to the Sum-Product algorithm [Kschischang, Frey and Loeliger 2001]. Convergence to an optimal solution is guaranteed for acyclic factor graphs only. The space and time complexity is $O(d^l)$, since each agent needs to store and optimise on the assignments propagated from its neighbours. As a consequence, the total number of messages is $O(tnl)$. The size of each message is $O(d)$, since it contains the value of all the possible assignments of each variable. Max-Sum has been widely extended, among the most notable variants we mention: bounded Max-Sum [Rogers et al. 2011; Rollon and Larrosa 2012], which bounds the solution quality by first turning cyclic graphs into spanning trees; Max-Sum_ADVP, which converges polynomially on acyclic graphs using a double-phase value propagation [Z. Chen, Deng and T. Wu 2017; Zivan and Peled 2012]; Max-Sum with damping [Cohen and Zivan 2017], which is guaranteed to converge to optimal solutions in weakly polynomial time. Max-Sum and its variants have been successfully deployed in disaster response [Delle Fave, Farinelli et al. 2012; Delle Fave, Rogers et al. 2012; Delle Fave, Xu et al. 2010; Pujol-Gonzalez, Jesus Cerquides, Meseguer et al. 2018; Ramchurn, Fischer et al. 2015; Ramchurn, Huynh, Ikuno et al. 2015; Ramchurn, Huynh, F. Wu et al. 2016; Stranders, Delle Fave et al. 2010; Stranders, Farinelli et al. 2009; Yedidsion, Zivan and Farinelli 2018] and various other domains, such as sensor networks, service-oriented computing, smart grid, and traffic management [Fioretto, Pontelli and Yeoh 2018].

GDBA *Generalised DBA* [Okamoto, Zivan, Nahon et al. 2016] is a class of incomplete, synchronous and search-based algorithms that extend the *Distributed Breakout Algorithm* (DBA) [Yokoo and Hirayama 1996] to solve DCOPs. These algorithms are not anytime, but can be made so with the Anytime Local Search framework[7] [Zivan, Okamoto and Peled 2014]. Moreover, they have polynomial space and time complexity, and local communication. The results reported in [Mahmud, Khan and Jennings 2020; Zivan, Okamoto and Peled 2014] suggest that the (N, NM, T) variant has similar performance to DSA-SDP (Page 23). Each GDBA variant has the same time and space complexity as DBA.

[7] In general, this framework can be used with any other incomplete and synchronous DCOP algorithm that is not anytime, such as DSA or Max-Sum.

CoCoA *Cooperative Constraint Approximation* [Leeuwen and Pawelczak 2017] is an incomplete, asynchronous and search-based algorithm based on: a one-step look ahead to assess how much an assignment might impact neighbours; a unique-first approach that selects an assignment only if it is a unique local optimum, and a global state machine that helps the algorithm to terminate. It has 2 limitations: the state machine forces the communication to be global, while the one-step look ahead does not make it possible to use the algorithm in dynamic environments (see Section 4.1.3 for a detailed explanation).

ACO_DCOP *Ant Colony Optimisation DCOP* [Z. Chen, T. Wu et al. 2018] was the first population-based DCOP algorithm, based on the homonymous swarm intelligence metaheuristic [Dorigo, Birattari and Stutzle 2006]. It is incomplete and asynchronous. Moreover, it is proven to be anytime, and has polynomial time and space complexity. Its drawback is the requirement of global communication, since ants (agents) lay pheromone to mark the promising paths (solution subspaces) that other members of the colony should follow (investigate).

COOPT *Coupled Oscillator OPTimisation* [Leite and Enembreck 2019b] is an incomplete, synchronous and search-based algorithm inspired by the synchronisation process in coupled oscillator networks. It is anytime, scalable and guaranteed to converge, and has a communication overhead similar to DSA-SDP. However, it uses a global communication model.

LSGA *Local Search Genetic Algorithm* [Z. Chen, L. Liu et al. 2020] constitutes a population-based hybrid framework that uses genetic operators to improve local explorations and avoid local optima. Solutions are constructed by global fitness functions, which detect and solve conflicts between partial candidate solutions. LSGA is incomplete and asynchronous, with linear time and space complexity. Although it can be used in conjunction with any search-based algorithm, it cannot provide anytime solutions.

AED *Anytime Evolutionary DCOP* [Mahmud, Choudhury et al. 2020] is an incomplete, synchronous and population-based algorithm based on the following evolutionary process. Each agent starts with a set of randomly generated candidate solutions (local population). In a series of iterations, the candidate solutions are first mutated (reproduction), then the least promising ones are discarded with a stochastic procedure. AED is anytime, has polynomial time and space complexity, and uses a local communication model. Its performance depends on tuning parameters (α and β), which define the balance between exploration and exploitation of the solution space. Consequently, it may require a non-trivial tuning phase to achieve the best results with specific problems. In fact, in [Mahmud, Khan and Jennings 2020], it (slightly) outperforms LSGA in only 2 out of 5 test suites.

DPSA *Distributed Parallel Simulated Annealing* [Mahmud, Khan and Jennings 2020] is an incomplete and asynchronous algorithm, with an approach based on both population and local search, similar to LSGA. It runs parallel instances of the Distributed Simulated Annealing algorithm [Arshad and Silaghi 2004], each of which is fine-tuned using *Cross-Entropy sampling* (CE) [Kroese, Taimre and Botev 2011, Section 9.7.3] to avoid convergence to local optima [Zivan, Okamoto and Peled 2014]. Like AED, it is anytime, has polynomial time and space complexity, is based on a local communication model, and depends on initial parameters (the CE vector θ). To alleviate the last point, the authors provide a pre-processing phase called *Greedy Baseline* (GB). Despite requiring fewer parameters than AED, DPSA still needs an initial tuning. In the scenarios we are interested in, first responders may be deployed at short notice, or the situation may evolve rapidly, hence prior knowledge about the search space of θ may not be available, and the GB phase may have poor performance.

2.3.2 Dynamic DCOP

In real-world scenarios, the environment may change over time. In disaster response, for instance, new information may become available after the start of the mission (e.g., an update of the number of victims, or new evacuation priorities), agents may fail, or more may be added to the system. The *Dynamic DCOP* (DynDCOP) is a generalisation of the DCOP capable of addressing such situations. Like the DCOP, it has deterministic agent behaviour, total agent knowledge, cooperative agent coordination, and deterministic environment behaviour. The only difference is that the environment evolution is dynamic.

Definition 2.4 A DynDCOP is a sequence of DCOPs, $\mathcal{D}_1, \dots, \mathcal{D}_\mathrm{T}$, where each $\mathcal{D}_t$ is the DCOP at time step t. The objective is to optimally solve $\mathcal{D}_t$, $\forall t \leq \mathrm{T}$.

Although it is assumed that agents have total knowledge about their current DCOP, they are unaware of how the problem may changes in the future. The naive solution to a DynDCOP is to solve each $\mathcal{D}_t$ with a DCOP algorithm. However, a clever algorithm design can exploit the *self-stabilising* property of dynamical systems [Schneider 1993] and minimise the number of iterations necessary to converge to a solution.

Definition 2.5 A DynDCOP is *self-stabilising* if and only if: a solution to $\mathcal{D}_t$ is obtained from a solution to $\mathcal{D}_{t-1}$ (*convergence*); a solution does not change after convergence (*closure*).

The DynDCOP is NP-hard, since it requires to solve a series of DCOPs. Dynamic environments pose a challenge to the DCOP research community [Barambones, Imbert and Moral 2021; Fioretto, Pontelli and Yeoh 2018; Leite, Enembreck and Barthes 2014; V. Lesser and Corkill 2014; Petcu 2007], to the extent that only four DynDCOP algorithms have been proposed to date[8]: the exact I(-BnB)-ADOPT and S-DPOP, and the incomplete SBDO and FMS.

[8]We exclude algorithms proposed for variants of the DynDCOP [Barambones, Imbert and Moral 2021, Section 4.4], as they are less general, or are extensions of DCOP algorithms, such as [Zivan, Yedidsion et al. 2015].

Algorithm	Error bound	Time complexity	Anytime	Space complexity per agent	Number of messages	Message size	Local communication
I(-BnB)-ADOPT	✓	$O(Td^n)$	×	$O(n + ld)$	$O(Td^n)$	$O(n)$	✓
S-DPOP	✓	$O(Td^w)$	×	$O(d^w)$	$O(Tn)$	$O(d^w)$	✓
FMS	×	$O(Ttd^l)$	×	$O(d^l)$	$O(Ttnl)$	$O(d)$	✓
SBDO	✓	$O(Td^n)$	✓	$O(n)$	$O(Td^n)$	$O(n)$	✓

Table 2.3 Characteristics of the DynDCOP algorithms proposed to date, where T denotes the total number of time steps, and the other variables are the same as those used in Table 2.2. The top side rows refer to exact algorithms, while those on the bottom side refer to incomplete algorithms.

I(-BnB)-ADOPT *Incremental anyspace (Branch-and-Bound) ADOPT* [Yeoh et al. 2015] is an exact, asynchronous and search-based algorithm that extends (Branch-and-Bound) ADOPT. Inspired by the *Multi-agent Organisation with Bounded Edit Distance* (MOBED) algorithm [Sultanik, Lass and Regli 2009], in order to minimise computations, it uses an incremental pseudo-tree reconstruction that reuses parts of the pseudo-tree of $\mathcal{D}_{t-1}$ to construct the pseudo-tree of $\mathcal{D}_t$. The computational complexity and communication requirements are the same as in ADOPT. However, having the anyspace property, this variant can use more computational space, when available, to improve the runtime.

S-DPOP *Self-stabilising DPOP* [Petcu and Faltings 2005b] is an extension of DPOP in which both the DFS pseudo-tree generation and the message propagation phases are re-executed whenever the DCOP formulation changes. The computational complexity and communication requirements are the same as those of DPOP. In addition, when the problem changes, S-DPOP stabilises in $O(t_d)$ utility messages (first propagation) and in $O(k)$ assignment (second propagation), where t_d is the depth of the pseudo-tree, and k is the number of utility functions.

SBDO *Support-Based Distributed Optimisation* [Billiau, Chang and Ghose 2012] is an incomplete, asynchronous and search-based algorithm that extends the *Support-Based Distributed Search* algorithm [Harvey, Chang and Ghose 2006] to multi-agent systems. In SBDO, each agent tries to send stronger arguments over time to influence its neighbours. Despite being anytime, SBDO has exponential runtime, being similar to SyncBB [Hirayama and Yokoo 1997]. [Billiau, Chang and Ghose 2014] proposed an extension to solve multi-objective DCOPs.

FMS *Fast Max-Sum* [Ramchurn, Farinelli et al. 2010] is the dynamic version of the incomplete Max-Sum, in which, at iteration t, solution stability is guaranteed by recalculating only the factors that changed between $\mathcal{D}_{t-1}$ and $\mathcal{D}_t$. Thus, the computational complexity does not change. FMS has been extended to provide error bounds on the solution [Macarthur, Farinelli et al. 2010], and to speed up the message propagation via branch-and-bound pruning [Macarthur, Stranders et al. 2011]. In addition, it can be further accelerated using the generic approaches proposed by [Z. Chen, Xingqiong Jiang et al. 2019; Khan, Tran-Thanh and Jennings 2018; Zaoad et al. 2021].

2.4 Our Chosen Approach

In the previous sections, we discussed state-of-the-art cooperative coordination approaches based on game theory, decision theory and constraint programming. As we mentioned in Section 1.3, the approach we choose is constraint programming, and specifically the DynDCOP. We give our technical motivations in the following paragraphs.

Why Not Game Theory Despite the advantages offered by the analytical approach, game theory has some limitations in our context. First of all, it is possible to define models applicable to specific problems, but it is not possible to define a general model to govern rational choice in interdependent situations [Zeng and Sycara 1996]. Moreover, it is often assumed perfect computational rationality [S. J. Gershman, Horvitz and Tenenbaum 2015; Lewis, Howes and S. Singh 2014], meaning that it is not necessary to perform calculations to find an acceptable solution in a set of possible outcomes. This is not realistic, since an agent can know its own space of solutions, but not that of others. Nonetheless, even if there was a shared solution space, knowing that a solution exists does not generally imply knowing what it is. Most CF models assume a limiting superadditive environment, since otherwise the computational complexity can be exponential [Sandholm, Larson et al. 1999, Section 2.2]. The computations can be distributed and hence decentralised (Requirement R1), but not always this process is balanced, in the sense that some agents will have to compute more than others. This issue is known as *coalition imbalance*, and can lead to situations where some agents have a dominant share of the capabilities. An *imbalanced* coalition relies more on its dominant agents, thus it is less fault-tolerant (Requirement R4). There are methods for partitioning the computational load almost equally, such as in auctions and blockchains, but at the cost of a major increase in complexity. In auctions, for instance, two open problems are efficiently minimising the communication overhead [Gerkey and Matarić 2004], and characterising the ability of agents to respond quickly to dynamic environments [Dias et al. 2006].

Why Not Decision Theory In decision theory, although there are algorithms able to handle large domain spaces successfully [Capitan et al. 2013], scalability remains a critical challenge, especially with POMDP formulations [Amato, Chowdhary et al. 2013; Seuken and Zilberstein 2008]. Popular models such as the Dec-POMDP [Bernstein et al. 2002] or the Networked Distributed POMDP [Nair, Varakantham et al. 2005] are NEXP-complete even for scenarios with just 2 agents, therefore they are limited to very small problems or subject to conditions with loss of generality [Kumar, Zilberstein and Toussaint 2011]. Other common models require noiseless channels or instantaneous communication between agents [Nair, Roth and Yohoo 2004; Pynadath and Tambe 2002; Roth, Simmons and Veloso 2005], which in general is not possible in decentralised environments like disaster response.

Why Constraint Programming For the above reasons, we regard the DynDCOP to be the most suitable for modelling distributed multi-agent cooperative coordination in dynamic environments. This model allows to consider the aspects that we aim for, such as autonomous decision-making, cooperation, and resilience (Section 1.3). Moreover:

- Communication strategies and problem solving are directly linked in (Dyn)DCOPs: the structure of the interaction graph can be exploited to generate efficient solutions;
- By definition, (Dyn)DCOPs focus on decentralisation, in particular because the constraints are decomposable, and agents can cooperatively define global solutions through local communication, as required in point-to-point environments such as those considered in disaster response;
- As we have seen in Section 2.3, there is a vast choice of (Dyn)DCOP models and algorithms, many of which have been successfully applied in disaster response, as well as many other real-world scenarios [Barambones, Imbert and Moral 2021; Jesus Cerquides et al. 2013; Fioretto, Pontelli and Yeoh 2018].

As anticipated by Figure 1.4, in the following chapters we will come to reduce both the CFSTP and the novel MARSC to the DynDCOP model. Using this reduction, we will create the first DynDCOP algorithm to be simultaneously anytime, efficient, convergent, and with local communication.

2.5 Problems Similar to the CFSTP

Many problems similar to the CFSTP have been studied to date [Dadvar and Habibian 2021; Juárez, Santos and Brizuela 2021; Murphy 2016; Murphy, Tadokoro and Kleiner 2016; Paraskevopoulos et al. 2017; Queralta et al. 2020; Rizk, Awad and Tunstel 2019; Seenu et al. 2020]. We mention below those that come closest to our target (Section 1.3). [Scerri et al. 2005] were the first to investigate time windows and interdependent simultaneous tasks in cooperative coordination, while [Service and Adams 2011; Vig and Adams 2006] applied the seminal work of [Shehory and Kraus 1998] to multi-robot systems. However, none of them not consider situations where task scheduling is required (Assumptions A1, A2, and A5). [Vig and Adams 2006] was extended in [Vig and Adams 2007] with task preemption and precedence relations, but no spatio-temporal constraints nor multiple possible locations per task (Assumptions A1 and A3). [Zlot 2006] defined a problem where tasks are decomposable in multiple ways, but do not have precedence relations (Assumption A2). [A. J. Singh, Dalapati and Animesh Dutta 2014; Su et al. 2018] studied multi-agent CF in dynamic environments, the former only with task priorities and the latter only with temporal constraints, while [Luo, Chakraborty and Sycara 2015] focused on multi-robot CF only with deadline constraints. [Ayari, Hadouaj and Ghedira 2017] proposed a dynamic, decentralised and efficient CF heuristic for MRTA problems with

priority constraints, however ignoring spatio-temporal constraints, task resources and multiple possible locations per task (Assumptions A1, A3 and A4). [Nelke, Okamoto and Zivan 2020; Tkach and Amador 2021] studied a CFSTP variant with task weights and soft deadlines, but without taking into account task precedences, time windows and multiple possible locations per task (Assumptions A1 − 3). [Bischoff et al. 2020] considered multi-agent CF with task precedences and no spatio-temporal constraints (Assumption A1). [Suslova and Fazli 2020] focused on multi-agent CF with time windows and ordering constraints, but in the special case where each task has the same weight and only one possible location, while coalitions are superadditive (i.e., the duration of a task depends only on the size of the assigned coalition). Similar to [Krausburg, Dix and Bordini 2021] (Section 2.1), [Präntare, Appelgren and Heintz 2021; Präntare and Heintz 2020] proposed a generalisation of the CSG problem able to capture task allocation with ordering constraints, but without spatio-temporal constraints, multiple possible locations per task, and task workloads (Assumptions A1, A3, and A4). [Arif 2021] applied evolutionary computation to multi-robot CF considering only homogeneous robots and no operational constraints.

Although they do not focus on CF, the following works have features of interest to our domain. [Barbulescu et al. 2010] considered task allocation for a team of agents with temporal, ordering and synchronisation constraints. As teams differ from coalitions (Figure 1.3), they do not consider routing constraints due to CF over time and in different locations. [Korsah 2011] investigated spatio-temporal constraints, task precedences and multiple possible locations per task, while [Godoy and Gini 2013; Nunes, McIntire and Gini 2017] studied problems with time windows, and spatial and precedence constraints. [Maoudj et al. 2015] studied MRTA with precedence constraints, robot capability constraints, and robot resource constraints. [Nanjanath and Gini 2010] applied combinatorial auction to MRTA in dynamic environments, with precedence constraints and time windows. [Whitbrook, Meng and Chung 2015, 2017, 2019] proposed distributed and resilient heuristics for real-time task allocation in multi-agent systems. [Feo-Flushing, Gambardella and Di Caro 2021] proposed decentralised, anytime and efficient algorithms for multi-robot routing and scheduling with heterogeneous agents, non-atomic tasks (i.e., preemptable), task workloads, and spatio-temporal constraints. Targeting large-scale multi-UAV flood response problems, [Ghassemi and Chowdhury 2021] defined a bigraph-based, scalable and online algorithm for MRTA in dynamic environments, with tasks deadlines and limited robot resources. [B. A. Ferreira, Petrović and Bogdan 2021] created a distributed metaheuristic for multi-robot scheduling with precedence constraints, based on a hierarchical task representation.

In the taxonomy of [Korsah, Stentz and Dias 2013], which extends that of [Gerkey and Matarić 2004], the CFSTP is defined as a *Cross-schedule Dependent Single-Task Multi-Robot Time-extended Assignment* (XD-ST-MR-TA) problem, where:

- ST means that each robot may work on at most one task at a time;
- MR implies that a task may require multiple robots, and thus coalition formation;

- TA indicates that each robot may have to work on multiple tasks according to some schedule;
- XD means that the schedule of an agent may also depend on the schedules of other agents. For instance, if we have 2 tasks and 2 agents, and the first task requires 2 agents, while the second task requires only one, then the first task can be executed only if the second has been completed, or no agent is working on it.

To date, the main approaches proposed to solve XD-ST-MR-TA problems utilise linear programming [Bogner et al. 2018; Koes, Nourbakhsh and Sycara 2005; Korsah 2011], automated negotiation [Krizmancic et al. 2020] and memetic algorithms [C. Liu and Kroll 2015]. However, either they do not produce anytime solutions [Krizmancic et al. 2020; C. Liu and Kroll 2015], or do not have theoretical properties [Bogner et al. 2018], or are based on models simpler than the CFSTP [Koes, Nourbakhsh and Sycara 2005; Korsah 2011]. Multi-agent approaches that solve similar problems typically make use of social insects [Amorim, Alves and Freitas 2020; Dos Santos and Bazzan 2011; Paulo R. Ferreira, Boffo and Bazzan 2007; Paulo Roberto Ferreira et al. 2010; Schwarzrock et al. 2018], automated negotiation [Gallud and Selva 2018; Godoy and Gini 2013; Nelke, Okamoto and Zivan 2020; Tkach and Amador 2021; Ye, M. Zhang and Sutanto 2013] and evolutionary computation [Zhou et al. 2020], but without considering the anytime property.

Although each of the works considered has interesting aspects, we note that no one meets all our requirements and assumptions. Consequently, given the importance of the CFSTP (Section 1.2), after filling main gaps in its literature in Chapters 4 and 5, we will conclude by presenting in Chapter 6 a generalisation able to capture more complex problems in disaster response, as well as in real-time CF problems in general.

Chapter 3

Background

In the previous chapter, we have seen that the CFSTP has been tackled using approaches from all fields of research reviewed [Baker et al. 2016; Nelke, Okamoto and Zivan 2020; Ramchurn, Farinelli et al. 2010; Ramchurn, Polukarov et al. 2010; Ramchurn, F. Wu et al. 2016; Tkach and Amador 2021]. As shown in Chapter 1, and as will also be evident below, the reason is that it generalises or shares similarities with classic combinatorial optimisation problems.

This chapter will serve as the basis for the rest of the thesis. In Section 3.1, we present an improved version of the constraint program of the CFSTP given by [Ramchurn, Polukarov et al. 2010]. More precisely, we extend the definition of coalition value, define the constraints with fewer and simpler equations, and introduce the concept of *solution degree*. In Section 3.2, we illustrate CFLA, the state-of-the-art CFSTP algorithm. In Chapter 4, we improve the design of CFLA, then we eliminate its limitations by formulating a novel algorithm. Finally, we give the original *Mixed Integer Program* (MIP) [Wolsey 2020] of the CFSTP in Section 3.3, which will be shown to be equivalent to a novel and significantly shorter program in Chapter 5.

3.1 The CFSTP Model

We first give our basic definitions (Section 3.1.1), then characterise coalition allocation and coalition values (Sections 3.1.2 − 3.1.3), and finally give the constraints (Section 3.1.4) and the objective function of the CFSTP (Section 3.1.5).

3.1.1 Basic Definitions

Let $\mathrm{V} = \{v_1, \dots, v_m\}$ be a set of m tasks and $\mathrm{A} = \{a_1, \dots, a_n\}$ be a set of n agents. Let L be the finite set of all possible task and agent locations. Time is denoted by $t \in \mathbb{N}$, starting at $t = 0$, and agents travel or complete tasks with a base time unit of 1. The time units needed by an agent to travel from one location to another are given by the function $\rho : \mathrm{A} \times \mathrm{L} \times \mathrm{L} \to \mathbb{N}$. Having A in

the domain of ρ allows to characterise different agent features (e.g., speed or type). Let l_v be the fixed location of task v, and let $l_a^t \in \mathrm{L}$ be the location of agent a at time t, where l_a^0 is its initial location and is known a priori. Each task v has a *demand* (γ_v, w_v) such that γ_v is the *deadline* of v, or the time until which agents can work on v [Nunes, Manner et al. 2017], and $w_v \in \mathbb{R}_{\geq 0}$ is the *workload* of v, or the amount of work required to complete v. We denote the location of agent a at time t by $l_a^t \in \mathrm{L}$, the times at which a starts and finishes working on task v by $s_a^v \in [0, \gamma_v]$ and $f_a^v \in [s_a^v, \gamma_v]$, respectively, and the *maximum problem time* by $t_{max} = \max_{v \in \mathrm{V}} \gamma_v$.

3.1.2 Coalition Allocations

A subset of agents $\mathrm{C} \subseteq \mathrm{A}$ is called a *coalition*. At time t, the rationale for allocating coalition C to task v is that C completes v in the fewest time units. An *agent allocation* is denoted by $\tau_t^{a \to v}$ and represents the fact that agent a works on task v at time t. The *set of all agent allocations* is denoted by:

$$\mathrm{T} = \left\{ \tau_t^{a \to v} \right\}_{a \in \mathrm{A},\, v \in \mathrm{V},\, t \in [0,\, t_{max}]} \tag{3.1}$$

and contains all possible agent allocations. A *coalition allocation* is denoted by $\tau_t^{\mathrm{C} \to v}$ and represents the fact that coalition C works on task v at time t. Given a set of agent allocations $\mathrm{T}' \subseteq \mathrm{T}$, and a time $t' \leq t_{max}$, the *set of coalition allocations corresponding to* T' over the time period $[0, t']$ is denoted by:

$$\Gamma(\mathrm{T}', t') = \left\{ \tau_t^{\mathrm{C} \to v} \mid \mathrm{C} = \left\{ a \mid \tau_t^{a \to v} \in \mathrm{T}' \right\}, t \leq t' \right\} \tag{3.2}$$

Furthermore, the *set of all coalition allocations* is denoted by:

$$\Gamma = \Gamma(\mathrm{T}, t_{max}) \tag{3.3}$$

Similar to T, Γ contains all possible coalition allocations. An agent allocation $\tau_t^{a \to v}$ is also denoted as a *singleton coalition allocation* $\tau_t^{\{a\} \to v}$.

3.1.3 Coalition Values

A subset of agents $\mathrm{C} \subseteq \mathrm{A}$ is called a *coalition*. Each coalition allocation $\tau_t^{\mathrm{C} \to v}$ has a *coalition value*, given by the function[1] $u : \mathrm{P(A)} \times \mathrm{V} \to \mathbb{R}_{\geq 0}$, where P(A) is the power set of A. The value of $u(\mathrm{C}, v)$ expresses how well the agents in C work together on v, and the workload w_v decreases by $u(\mathrm{C}, v)$ at each time.

[1] In cooperative game theory, this is called a *characteristic function* [Chalkiadakis, Elkind and Wooldridge 2012, Section 2.1].

3.1.4 Constraints

There are 3 types of constraints: structural, temporal and spatial. Structural constraints require that each task v can be allocated to only one coalition at a time. This is characterised by the following sets:

$$\forall v \in \mathrm{V}, \Gamma_v = \left\{ \Gamma' \subseteq \Gamma \,:\, \tau_t^{\mathrm{C}_1 \to v}, \tau_t^{\mathrm{C}_2 \to v} \in \Gamma' \Rightarrow \mathrm{C}_1 = \mathrm{C}_2 \right\} \tag{3.4}$$

With an abuse of notation, we write $\tau_t^{\mathrm{C} \to v} \in \Gamma_v$ to indicate that $\tau_t^{\mathrm{C} \to v}$ belongs to an unspecified set of Γ_v. Temporal constraints require that each task v can be completed only by its deadline γ_v. This is characterised by the function $\Delta \,:\, \mathrm{V} \times \Gamma \to \{0, 1\}$, defined as follows:

$$\Delta(v, \Gamma) = \begin{cases} 1, & \text{if } \exists\, t \leq \gamma_v \,:\, \sum_{t' \leq t,\, \tau_{t'}^{\mathrm{C} \to v} \in \Gamma_v} u(\mathrm{C}, v) \geq w_v \\ 0, & \text{otherwise} \end{cases} \tag{3.5}$$

Equation 3.5 utilises Γ_v (Equation 3.4) to count only coalition allocations that satisfy the structural constraints. Spatial constraints require that an agent will not start working on a task before reaching it. This is characterised as follows:

$$\forall a \in \mathrm{A},\ \forall v \in \mathrm{V}, \forall t \leq \gamma_v,\ s_a^v \geq t + \rho(a, l_a^t, l_v) \tag{3.6}$$

$$\forall a \in \mathrm{A}, \forall v_1, v_2 \in \mathrm{V},\ f_a^{v_1} + \rho(a, l_{v_1}, l_{v_2}) \leq s_a^{v_2} \tag{3.7}$$

A set of agent allocations $\mathrm{T}' \subseteq \mathrm{T}$ is called *legal* if it exists a time $t' \leq t_{max}$ such that $\Gamma(\mathrm{T}', t')$ satisfies Equation 3.5. A set of coalition allocations $\Gamma' \subseteq \Gamma$ that satisfies Equations $3.5 - 3.7$ is called *feasible*. Consequently, at time t, if $\tau_t^{\mathrm{C}_1 \to v_1}$ and $\tau_t^{\mathrm{C}_2 \to v_2}$ are feasible coalition allocations and $l_{v_1} \neq l_{v_2}$, then $\mathrm{C}_1 \cap \mathrm{C}_2 = \emptyset$.

There are no synchronisation constraints [Nunes, Manner et al. 2017]: when a task v is allocated to a coalition C, each agent $a \in \mathrm{C}$ starts working on v as soon as it reaches l_v. Hence, v is completed by a temporal sequence of subcoalitions of C.

3.1.5 Objective Function

The objective function of the CFSTP is to find a set of feasible coalition allocations that maximises the number of completed tasks:

$$\arg \max_{\substack{\Gamma' \subseteq \Gamma \\ \Gamma' \text{feasible}}} \sum_{v \in \mathrm{V}} \Delta(v, \Gamma') \tag{3.8}$$

To solve Equation 3.8, an exhaustive search may require to verify all the possible coalition allocations until t_{max}. Consequently, the time complexity of searching for an optimal solution is:

$$\mathrm{O}\left(|\mathrm{V}|! \cdot 2^{|\mathrm{A}|} \cdot t_{max} \right) \tag{3.9}$$

Algorithm 3.1: getLegalAgentAllocations (Phase 1 of CFLA)

Input: time t

$\text{Leg}_t \leftarrow \emptyset$ // the set of legal agent allocations at time t
for $a \in \text{A}^t_{free}$ **do** // for each free agent a
  for $v \in \text{V}_{unc}$ **do** // for each uncompleted task v
    if $t + \rho(a, l^t_a, l_v) \leq \gamma_v$ **then** // if a can reach v at t by γ_v
      $\text{Leg}_t \leftarrow \text{Leg}_t \cup \{\tau^{a \to v}_{t'}\}_{t+\rho(a,l^t_a,l_v) \leq t' \leq \gamma_v}$
return Leg_t

A set of feasible coalition allocations $\Gamma' \subseteq \Gamma$ is called a *solution with degree k* if $\sum_{v \in \text{V}} \Delta(v, \Gamma') = k$. Hence, an argument of Equation 3.8 is a solution with the highest degree. Since it generalises the TOP (Figure 1.4), the CFSTP is NP-hard [Papadimitriou 1993].

3.2 The CFLA Algorithm

In this section, we report the concept of CFLA and the procedures of which it is composed. CFLA has 4 phases, but [Ramchurn, Polukarov et al. 2010, Section 6] describes them in 3 procedures. For readability purposes, we describe them in 4.

3.2.1 The Concept of CFLA

CFLA is a centralised, anytime and greedy algorithm that solves Equation 3.8 by maximising the working time of agents and minimising the time required by coalitions to complete tasks. It is divided into 4 phases:

1. Defining the legal agent allocations (Section 3.2.2);
2. For each task v, choosing the best coalition C (Section 3.2.3);
3. For each task v, doing a 1-step look-ahead (Section 3.2.4) to define its *degree* δ_v, or the number of tasks that can be completed after the completion of v;
4. At each time $t \leq t_{max}$, allocating a task not yet completed and with the highest degree (Section 3.2.5).

3.2.2 Phase 1: Defining the Legal Agent Allocations

At time t, Algorithm 3.1 determines which free agents[2] A^t_{free} can reach which uncompleted tasks V_{unc} by their deadlines. The resulting set of legal agent allocations is denoted by Leg_t.

[2]That is, agents who neither are travelling to nor working on a task.

Algorithm 3.2: ECF (Phase 2 of CFLA)

Input: task v, a set of legal agent allocations Leg_t

1 $\text{C}_v^* \leftarrow \emptyset$ // an ECF coalition

2 Using Leg_t, define $\Gamma(\text{T}', \gamma_v)$ as in Equation 3.2

// minimise |C|, where C can complete v by its deadline γ_v

3 Find $minsize = \min_{\tau_t^{\text{C}\to v} \in \Gamma(\text{T}', \gamma_v)} |\text{C}|$ where $\sum_{\tau_{t'}^{\text{C}\to v},\, t \le t' \le \gamma_v} u(\text{C}, v) \ge w_v$

4 $t_v^{\min} \leftarrow \gamma_v$

// loop through all possible coalitions of size *minsize*

5 **for** $\tau_t^{\text{C}\to v} \in \Gamma(\text{T}', \gamma_v)$ where $|\text{C}| = minsize$ **do**

6 **if** $\sum_{\tau_{t'}^{\text{C}\to v},\, t \le t' \le \gamma_v} u(\text{C}, v) \ge 0$ **then** // C can complete v by its deadline γ_v

// minimum time at which C can complete v

7 $t_{\text{minmax}} \leftarrow \min_{t_{\max}} \left(w_v - \sum_{\tau_{t'}^{\text{C}\to v},\, t \le t' \le t_{\max}} u(\text{C}, v) \right)$

8 **if** $t_{\text{minmax}} < t_v^{\min}$ **then** // check if C is the new best coalition

9 $t_v^{\min} = t_{\text{minmax}}$

10 $\text{C}_v^* \leftarrow \text{C}$

11 **return** C_v^*

3.2.3 Phase 2: Selecting the Best Coalition for Each Task

Given a task v and a set of legal agent allocations Leg_t (computed by Algorithm 3.1), Algorithm 3.2 returns an *Earliest Completion First* (ECF) coalition C_v^* that can be allocated to v. In other words, C_v^* is chosen such that the completion time of v is minimised and the agent in C_v^* become free to work on the remaining uncompleted tasks as soon as possible. This algorithm is myopic [Ramchurn, Polukarov et al. 2010, Section 6.1], since it does not consider the system status after that C_v^* would complete v. For this reason, Ramchurn et al. introduced the concept of task degree (Section 3.2.1) and developed the look-ahead technique presented below.

3.2.4 Phase 3: Defining the Degree of Each Task

Given a task v, Algorithm 3.3 performs a brute-force search to define its degree δ_v (Section 3.2.1). At line 8, with a procedure similar to line 5 in Algorithm 3.2, it checks how many tasks can be completed after the completion of v. Hence, Algorithm 3.3 assigns a score to each coalition allocation selected by Phase 2 (Section 3.2.3) for each currently uncompleted task. These scores are then used by Phase 4 (Section 3.2.5) to choose the next task to execute.

3.2.5 Phase 4: Overall Procedure of CFLA

Algorithm 3.4 shows the overall procedure. The repeat-until loop runs until all tasks are completed, or until the maximum problem time is expired (line 22). At each time t, the set of legal agent allocations is updated (line 8), and a task allocation is defined (Lines $9-18$). If it is not possible to allocate other tasks, the algorithm stops early (line 19).

Algorithm 3.3: lookAhead (Phase 3 of CFLA)

Input: task v, an ECF coalition C_v^* of v, current solution Γ'

$\delta_v \leftarrow 0$ // the degree of v
$f_v \leftarrow$ time at which C_v^* completes v
for $v_2 \in V_{unc} \setminus \{v\}$ **do**
  $A_{free}^{f_v} \leftarrow$ agents that are free at f_v // derived from C_v^* and Γ'
  $A^{d_{v_2}} \leftarrow$ select from $A_{free}^{f_v}$ all agents that can reach v_2 by d_{v_2}
  $i \leftarrow 1$
  while $i \leq |A^{d_{v_2}}|$ **do**
    for $C \in$ all combinations of i agents in $A^{d_{v_2}}$ **do**
      if $\sum_{\tau_t^{C' \to v} \in \Gamma_v,\, C' \subseteq C,\, t \in [f_v, d_{v_2}]} u(C, v) \geq w_v$ **then** // if C can complete v_2
        $\delta_v \leftarrow \delta_v + 1$
        $i \leftarrow |A^{d_{v_2}}|$ // break external loop too
        break
    $i \leftarrow i + 1$
return δ_v

3.3 The Original Mixed Integer Program of the CFSTP

As opposed to [Ramchurn, Polukarov et al. 2010, Section 5], for readability purposes, we begin by introducing the decision variables, then we formulate and explain the constraints, and finally give the objective function. To be consistent with the notation introduced so far, we rename some variables and constraints.

3.3.1 Decision Variables

Let $L_V \subseteq L$ denote the set of all task locations. There are 4 sets of binary variables and 1 set of integer variables:

$$\forall v \in V,\ \forall t \leq t_{max},\ \forall a \in A,\ x_{v,t,a} \in \{0,1\} \tag{3.10}$$

$$\forall v \in V,\ \forall t \leq t_{max},\ \forall C \subseteq A,\ x_{v,t,C} \in \{0,1\} \tag{3.11}$$

$$\forall v \in V,\ y_v \in \{0,1\} \tag{3.12}$$

$$\forall a \in A,\ \forall l_1 \in L,\ \forall l_2 \in L_V,\ r^a_{l_1,l_2} \in \{0,1\} \tag{3.13}$$

$$\forall v \in V,\ \forall a \in A,\ \lambda^v_a \in \{0, \ldots, t_{max}\} \tag{3.14}$$

where: $x_{v,t,a} = 1$ (resp. $x_{v,t,C} = 1$) if agent a (resp. coalition C) works on task v at time t, and 0 otherwise; $y_v = 1$ if task v is completed, and 0 otherwise; $r^a_{l_1,l_2} = 1$ if agent a travels from location $l_1 \in L$ to location $l_2 \in L_V$, and 0 otherwise, and λ^v_a is the time at which agent a starts working on task v. It is assumed that the allocation process starts at $t = 1$. Hence, $\lambda^v_a = 0$ indicates that agent a does not work on task v. In the original formulation, y_v is δ_v, $x_{v,t,C}$ is $\tau_t^{C \to v}$, and λ^v_a is s^v_a.

Algorithm 3.4: Overall procedure (Phase 4 of CFLA)

Input: tasks V, task demands $(\gamma_v, w_v)_{v \in V}$, agents A, locations L, travel function ρ, coalition value function u

```
1  t ← 0
2  Γ′ ← ∅ // a solution
3  V_unc ← V // uncompleted tasks
4  repeat
5      δ_max ← 0 // maximum task degree
6      v* ← NIL // next task to allocate
7      C* ← ∅ // coalition to which v* is allocated
8      Leg_t ← getLegalAgentAllocations(t) // Algorithm 3.1
9      for v ∈ V_unc do
10         C*_v ← ECF(v, Leg_t) // Algorithm 3.2
11         δ_v ← lookAhead(v, C*_v, Γ′) // Algorithm 3.3
12         if δ_v > δ_max then
13             δ_max ← δ_v
14             v* ← v
15             C* ← C*_v
16     if v* ≠ NIL and C* ≠ ∅ then
17         Add to Γ′ the allocation of C* to v*
18         V_unc ← V_unc \ {v*}
19     if A^t_free = A then // all agents are free
20         break
21     t ← t + 1
22 until V_unc = ∅ or t > t_max
23 return Γ′
```

3.3.2 Constraints

There are 4 types of constraints: *completion*, *temporal*, *spatial*, and *linking*. In the original formulation, the temporal constraints are called *deadline* constraints, while the spatial constraints are called *starting time, routing and service consistency* constraints.

The following equations use the Big-M method [Griva, Nash and Sofer 2009]: 3.16, 3.19, 3.20, 3.24, 3.28, 3.29, 3.32, 3.33, and 3.35. This method introduces an arbitrarily large positive constant M to artificially penalise the violation of the constraints involved.

Completion Constraints The work done for each task v is at least equal to its workload w_v if the task is completed, and 0 otherwise:

$$\forall v \in V, \sum_{t \leq t_{max}} \sum_{C \subseteq A} x_{v,t,C} \cdot u(C, v) \geq y_v \cdot w_v \tag{3.15}$$

$$\forall v \in V, \sum_{t \leq t_{max}} \sum_{C \subseteq A} x_{v,t,C} \leq M \cdot y_v \tag{3.16}$$

Moreover, at each time, at most one coalition can work on each task:

$$\forall v \in \mathrm{V},\ \forall t \leq t_{max},\ \sum_{\mathrm{C} \in \mathrm{A}} x_{v,t,\mathrm{C}} \leq y_v \tag{3.17}$$

Temporal Constraints Each task can only be completed by its deadline:

$$\forall v \in \mathrm{V},\ \forall a \in \mathrm{A},\ \lambda_a^v + \sum_{t \leq t_{max}} x_{v,t,a} \leq \gamma_v \tag{3.18}$$

Since it is assumed that there are no allocations at time $t = 0$, it follows from Equations 3.15 and 3.16 that if a task is not completed, the left-hand side of Equation 3.18 is 0.

Spatial Constraints For each agent a and task v, a can start to work on v after finishing work on a previous task, or after reaching the task location l_v from its initial location l_0^a:

$$\forall a \in \mathrm{A},\ \forall l \in \mathrm{L_V} \cup \left\{ l_0^a \right\},\ \forall v \in \mathrm{V},\ \rho\left(a, l, l_v\right) \leq \lambda_a^v + \mathrm{M} \cdot \left(1 - r^a_{l_a, l_v}\right) \tag{3.19}$$

$$\forall a \in \mathrm{A},\ \forall v_1, v_2 \in \mathrm{V},\ \lambda_a^{v_1} + \sum_{t \leq t_{max}} x_{v_1,t,a} + \rho\left(a, l_{v_1}, l_{v_2}\right) \leq \lambda_a^{v_2} + \mathrm{M} \cdot \left(1 - r^a_{l_{v_1}, l_{v_2}}\right) \tag{3.20}$$

For each agent a and time t, a can work on at most one task during t:

$$\forall a \in \mathrm{A},\ \forall t \leq t_{max},\ \sum_{v \in \mathrm{V}} x_{v,t,a} \leq 1 \tag{3.21}$$

If an agent starts working on a task at time t, then variables $x_{v,t,a}$ and $x_{v,t-1,a}$ must have different values:

$$\forall a \in \mathrm{A},\ \forall v \in \mathrm{V},\ \forall t \in [1, t_{max}],\ 1 - 2 \cdot \left| t - \lambda_a^v \right| \leq x_{v,t,a} - x_{v,t-1,a} \tag{3.22}$$

For each agent a and task v, a changes the status of its service on v (e.g., *not working* to *working* or vice versa) at most twice:

$$\forall a \in \mathrm{A},\ \forall v \in \mathrm{V},\ \sum_{t \in [1, t_{max}]} \left| x_{v,t,a} - x_{v,t-1,a} \right| \leq 2 \tag{3.23}$$

$$\begin{gathered} \forall a \in \mathrm{A},\ \forall v_1, v_2 \in \mathrm{V},\ \forall t \in [1, t_{max}], \\ 1 - \left| t - \lambda_a^{v_2} - \rho\left(a, l_{v_1}, l_{v_2}\right) \right| - \mathrm{M} \cdot \left(1 - r^a_{l_{v_1}, l_{v_2}}\right) \leq \left| x_{v_1,t,a} - x_{v_1,t-1,a} \right| \end{gathered} \tag{3.24}$$

An agent cannot leave and reach the same task location, it can only reach a new task location from exactly one previous location, and it can only leave a location to reach exactly one task location:

$$\forall a \in \mathrm{A},\ l \in \mathrm{L_V},\ r^a_{l,l} = 0 \tag{3.25}$$

$$\forall a \in \mathrm{A},\ \forall l \in \mathrm{L_V} \cup \left\{ l_0^a \right\},\ \forall v_1 \in \mathrm{V},\ r^a_{l,l_{v_1}} + \sum_{v_2 \in \mathrm{V} \setminus \{v_1\}} r^a_{l_{v_1}, l_{v_2}} \leq 1 \tag{3.26}$$

$$\forall a \in \mathrm{A},\ \forall l \in \mathrm{L_V} \cup \left\{ l_0^a \right\},\ \sum_{v \in \mathrm{V}} r_{l,l_v}^a \leq 1 \tag{3.27}$$

$$\forall v_2 \in \mathrm{V},\ \forall a \in \mathrm{A},\ \sum_{v_1 \in \mathrm{V} \setminus \{v_2\}} r_{l_{v_1},l_{v_2}}^a \leq \mathrm{M} \cdot \sum_{t \leq t_{max}} x_{v_2,t,a} \tag{3.28}$$

$$\forall v_1 \in \mathrm{V},\ \forall a \in \mathrm{A},\ \sum_{v_2 \in \mathrm{V} \setminus \{v_1\}} r_{l_{v_1},l_{v_2}}^a \leq \mathrm{M} \cdot \sum_{t \leq t_{max}} x_{v_1,t,a} \tag{3.29}$$

Linking Constraints An agent reaches a task only to complete it:

$$\forall v \in \mathrm{V},\ \sum_{a \in \mathrm{A}, l \in \mathrm{L_V} \cup \left\{ l_0^a \right\}} r_{l,l_v}^a \geq y_v \tag{3.30}$$

A coalition C works on task v and at time t only if every agent $a \in \mathrm{C}$ works on v at t:

$$\forall \mathrm{C} \subseteq \mathrm{A},\ \forall v \in \mathrm{V},\ \forall t \leq t_{max},\ \sum_{a \in \mathrm{C}} x_{v,t,a} \geq |\mathrm{C}| \cdot x_{v,t,\mathrm{C}} \tag{3.31}$$

If task v is allocated to agent a, then a works on v for at least 1 unit of time:

$$\forall a \in \mathrm{A},\ \forall v \in \mathrm{V},\ \lambda_a^v \leq \mathrm{M} \cdot \sum_{t \leq t_{max}} x_{v,t,a} \tag{3.32}$$

$$\forall a \in \mathrm{A},\ \forall v \in \mathrm{V},\ \sum_{t \leq t_{max}} x_{v,t,a} \leq \mathrm{M} \cdot \lambda_a^v \tag{3.33}$$

If task v is allocated to agent a, then a works on v after reaching l_v from another task location or from its initial location l_0^a:

$$\forall a \in \mathrm{A},\ \forall v \in \mathrm{V},\ \lambda_a^v \geq \sum_{l \in \mathrm{L_V} \cup \left\{ l_0^a \right\}} r_{l,l_v}^a \tag{3.34}$$

$$\forall a \in \mathrm{A},\ \forall v \in \mathrm{V},\ \lambda_a^v \leq \mathrm{M} \cdot \sum_{l \in \mathrm{L_V} \cup \left\{ l_0^a \right\}} r_{l,l_v}^a \tag{3.35}$$

Each agent can work on at most one task at a time, and cannot work at time $t = 0$:

$$\forall a \in \mathrm{A},\ \forall t \leq t_{max},\ \sum_{v \in \mathrm{V}} x_{v,t,a} \leq 1 \tag{3.36}$$

$$\forall a \in \mathrm{A},\ \forall v \in \mathrm{V},\ x_{v,0,a} = 0 \tag{3.37}$$

Equations 3.23, 3.31 and 3.37 also imply that if a coalition C works on a task, the service status of each $a \in \mathrm{C}$ switches from 0 to 1 at the beginning of the work, and from 1 to 0 at the end of it.

3.3.3 Objective Function

The objective of the CFSTP is to maximise the number of tasks completed:

$$\max \sum_{v \in \mathrm{V}} y_v \text{ subject to Equations } 3.15 - 3.37 \tag{3.38}$$

To solve the MIP defined by Equation 3.38, we first need to create all decision variables and constraints. Creating the decision variables of Equation 3.11 requires to list all $\mathcal{L}$-tuples over P(A), where $\mathcal{L} = |\mathrm{V}| \cdot t_{max}$, with a worst-case time and space complexity of:

$$\mathrm{O}\left(\left(2^{|\mathrm{A}|}\right)^{\mathcal{L}}\right) = \mathrm{O}\left(2^{|\mathrm{A}| \cdot |\mathrm{V}| \cdot t_{max}}\right) \tag{3.39}$$

Creating the remaining decision variables and constraints does not take more time and space than Equation 3.39. Hence, implementing and solving the above MIP with optimisation software packages such as CPLEX or GLPK may require an exponential amount of time and space, followed in the worst case by a factorial amount of time (Equation 3.9).

Chapter 4

Anytime and Efficient Multi-agent Coordination for Disaster Response

In this chapter, we begin by presenting the *Coalition Formation with improved Look-Ahead* (CFLA2), an extension of the CFLA algorithm (Section 3.2). Despite having better performance, CFLA2 keeps the design limitations of CFLA, which is why we propose the novel CTS algorithm in Section 4.2. We compare CTS with CFLA and CFLA2 in Section 4.3, and evaluate its performance with a well-established testbed in Section 4.4.

4.1 Coalition Formation with Improved Look-ahead

We give a more detailed formulation of Phase 2 in Section 4.1.1, and describe an improved Phase 3 in Section 4.1.2, which constitutes the novelty of CFLA2. Finally, we list the design limitations of both CFLA and CFLA2 in Section 4.1.3.

4.1.1 Forming Coalitions with Legal Agent Allocations

To minimise both the size of C_v^* and the time at which it completes task v, Algorithm 4.1 iterates from the smallest to the largest possible coalition size (line 5), and through all possible coalitions of each size (line 6). When the procedure finds a coalition C that can complete v by its deadline γ_v (line 7), then $|C|$ is the minimum size of the coalitions that can complete v. Hence, C_v^* is identified among the coalitions with size $|C|$ (lines $8-11$). The summations at lines $7-8$ capture the workload done by the coalition allocations defined from the asynchronous arrivals (Section 3.1.4) of the agents in C to the location of v.

Algorithm 4.1: ECF (more detailed Phase 2 of CFLA2)

Input: task v, a set of legal agent allocations Leg_t

1 $\text{A}_v^t \leftarrow$ define from Leg_t the agents that can reach v at t by γ_v
2 $\text{C}_v^* \leftarrow \emptyset$ `// the ECF coalition`
3 $t_v^* \leftarrow \gamma_v + 1$ `// time at which` C_v^* `completes` v
4 $i \leftarrow 1$
5 **while** $i \leq |\text{A}_v^t|$ and $\text{C}_v^* = \emptyset$ **do**
6 **for** $\text{C} \in$ all combinations of i agents in A_v^t **do**
7 **if** $\sum_{\tau_{t'}^{\text{C}\to v} \in \Gamma_v,\ \text{C}' \subseteq \text{C},\ t' \in [t, \gamma_v]} u(\text{C}, v) \geq w_v$ **then**
8 $t_{minmax} \leftarrow \min_{t_{max}} \left(w_v - \sum_{\tau_{t'}^{\text{C}\to v} \in \Gamma_v,\ \text{C}' \subseteq \text{C},\ t' \in [t, t_{max}]} u(\text{C}, v) \right)$
9 **if** $t_{minmax} < t_v^*$ **then**
10 $t_v^* \leftarrow t_{minmax}$
11 $\text{C}_v^* \leftarrow \text{C}$
12 $i \leftarrow i + 1$
13 **return** C_v^*

Unlike the original formulation (Algorithm 3.2), Algorithm 4.1 clarifies that the minimum coalition size is determined by iterating through subsets of the combinations[1] of A_v^t, which is the set of free agents that at time t can reach v by γ_v.

4.1.2 More Effective Task Degrees

Algorithm 4.2 differs from the original look-ahead phase (Algorithm 4.2) in 2 points. First, it only considers uncompleted tasks that have a deadline greater or equal to γ_v (line 4), which prevents from counting tasks that can be completed before the completion of v. This is because, as defined in Section 3.2.1, δ_v must represent the number of tasks that can be completed only after the completion of v. Second, at line 11, δ_v is not just incremented by 1, but also by $1 - \eta_{v_2}$, where η_{v_2} is the rescaling[2] of w_{v_2} to the range $[w_{min}, w_{max}]$, with w_{min} and w_{max} being respectively the minimum and maximum task workloads: $\eta_{v_2} = \left(w_{v_2} - w_{min}\right) / (w_{max} - w_{min})$. Hence, δ_v is also a measure of how much total workload remains after the completion of v. Maximising δ_v (line 12 in Algorithm 3.4) leads to the remaining tasks with the smallest workloads, which increases the probability of completing more.

4.1.3 Analysis and Discussion

Algorithm 3.1 iterates through all free agents and uncompleted tasks. Assuming that line 4 requires constant time, the time complexity is $\bar{a} = \text{O}(|\text{A}| \cdot |\text{V}|)$.

[1] To date, the most efficient technique to enumerate all such combinations is the Gray binary code [Knuth 2005, Section 7.2.1.1].

[2] Also known as min-max scaling or min-max normalisation.

Algorithm 4.2: lookAhead (improved Phase 3 of CFLA2)

Input: task v, its ECF coalition C^*_v, the set of all agent allocations T

1 $\delta_v \leftarrow 0$ `// the degree of task` v
2 $f_v \leftarrow$ time at which C^*_v completes v
3 **for** $v_2 \in V_{unc} \setminus \{v\}$ **do**
4 **if** $d_{v_2} \geq \gamma_v$ **then**
5 $A^{f_v}_{free} \leftarrow$ agents that are free at f_v `// derived from` C^*_v `and` T
6 $A^{d_{v_2}} \leftarrow$ select from $A^{f_v}_{free}$ the agents that can reach v_2 by d_{v_2}
7 $i \leftarrow 1$
8 **while** $i \leq |A^{d_{v_2}}|$ **do**
9 **for** C $\in$ all combinations of i agents in $A^{d_{v_2}}$ **do**
 `// if` C `can complete` v_2
10 **if** $\sum_{\tau_t^{C' \to v} \in \Gamma_v,\, C' \subseteq C,\, t \in [f_v, d_{v_2}]} u(C, v) \geq w_v$ **then**
11 $\delta_v \leftarrow \delta_v + 1 + (1 - \eta_{v_2})$
12 $i \leftarrow |A^{d_{v_2}}|$ `// break external loop too`
13 **break**
14 $i \leftarrow i + 1$
15 **return** δ_v

Algorithm 4.1 iterates (line 5) from coalition size 1 to $|A^t_v|$, where A^t_v is the set of agents that can reach task v at time t. This needs $O(|A|)$ time in case $A^t_v = A$. For each $i \leq |A^t_v|$, examining the set of all possible coalitions of size i (line 6) requires $O(2^{|A|})$ time. Assuming that line 8 takes $O(t_{max})$ time, the total time complexity is $\bar{b} = O(|A| \cdot 2^{|A|} \cdot t_{max})$.

Algorithm 4.2 iterates through all uncompleted tasks, which requires $O(|V|)$ time, while line 8 is computationally identical to line 5 in Algorithm 4.1. Hence, the time complexity is $\bar{c} = O(|V| \cdot 2^{|A|})$.

Algorithms 3.2 and 3.3 have the same time complexity of Algorithms 4.1 and 4.2, respectively. As it uses Phases $1 - 3$, Algorithm 3.4 has a time complexity of:

$$O\left(t_{max} \cdot \left(\bar{a} + |V| \cdot (\bar{b} + \bar{c})\right)\right) = O\left((t_{max} \cdot |V|)^2 \cdot 2^{|A|}\right) \tag{4.1}$$

Therefore, the runtime of CFLA2 is quadratic in the number of tasks and exponential in the number of agents, which makes it not suitable for systems with limited resources or real-time applications (Section 1.2). Other limitations are:

1. It can allocate at most one task per time [Ramchurn, Polukarov et al. 2010, Section 7]. More formally, at each time t, the best-case guarantee of CFLA2 is to find a solution with degree $k = 1$;

2. In general, greedily allocating a task with the highest degree now does not ensure to allocate all uncompleted tasks in future. This is particularly relevant in dynamic environments, where there is no certainty of having further tasks to be completed (Section 1.2);

3. The more tasks can be grouped by degree, the more the look-ahead technique becomes a costly random choice. In other words, at time t, if some tasks $V' \subseteq V$ have all maximum degree, then Algorithm 3.4 selects v^* randomly from V'. Hence, the larger V' is, the less relevant Algorithm 4.2 becomes;

4. In Algorithm 3.4, all tasks have the same weight. That is, tasks with earlier deadlines may not be allocated before tasks with later deadlines. This is independent of the order in which uncompleted tasks are elaborated (line 9), since the computation of δ_{max} (line 11) would not be affected.

To overcome the limitations of CFLA2, in the next section we present a CFSTP algorithm that is anytime, efficient and with convergence guarantee, both in static and dynamic environments.

4.2 Cluster-based Task Scheduling

Cluster-based Task Scheduling (CTS) algorithm operates at the agent level, rather than at the coalition level. Using the terminology of [Gerkey and Matarić 2004], we can say that it decomposes a Single-Task Multi-Robot problem into a sequence of Single-Task Single-Robot problems. It is divided into the following two phases:

1. For each free agent a, associate a with an uncompleted task v such that v is the closest to a and deadline γ_v is minimum;

2. For each uncompleted task v, allocate v to a coalition C such that $|C|$ is minimum and each agent $a \in C$ has been associated with v in Phase 1.

Algorithm 4.3 is used in Phase 1, while Algorithm 4.4 enacts both phases. We describe them in Sections 4.2.1 and 4.2.2, respectively.

4.2.1 Selecting the Best Task for Each Agent

Given a time t and an agent a, Algorithm 4.3 returns the uncompleted task v that is allocable, the most urgent and closest to a. By *allocable* we mean that a can reach v before deadline γ_v, while *most urgent* means that v has the earliest deadline. The algorithm prioritises unallocated tasks, that is, it first tries to find a task to which no agents are travelling, and on which no agents are working ($v_a^t[0]$). Otherwise, it returns an already allocated but still uncompleted task such that a can reach it and contribute to its completion ($v_a^t[1]$). This ensures that a becomes free only when no other tasks are uncompleted and allocable to a and.

Algorithm 4.3: getTaskAllocableToAgent (used in Phase 1 of CTS)

Input: time t, agent a

1 $v_a^t \leftarrow (\text{NIL}, \text{NIL})$ // array in which $v_a^t[0]$ (resp. $v_a^t[1]$) is the unallocated (resp. allocated but still uncompleted) task allocable to agent a at time t
2 $t_{min} \leftarrow (t_{max}+1, t_{max}+1)$ // array in which $t_{min}[0]$ (resp. $t_{min}[1]$) defines the time units required by agent a to reach $v_a^t[0]$ (resp. $v_a^t[1]$)
3 $t_{min} \leftarrow (t_{max}+1, t_{max}+1)$ // array in which $t_{min}[0]$ (resp. $t_{min}[1]$) is the deadline of $v_a^t[0]$ (resp. $v_a^t[1]$)
4 **for** $v \in V$ **do** // for each uncompleted task
5 $i \leftarrow 0$ // v is unallocated
6 **if** other agents are travelling to or working on v **then**
7 $i \leftarrow 1$ // v is allocated but still uncompleted
8 $t_{arr} \leftarrow t + \rho(a, l_a^t, l_v)$
9 **if** $t_{arr} \leq \gamma_v$ and $t_{arr} < t_{min}[i]$ and $\gamma_v < t_{min}[i]$ **then** // a can reach v by γ_v and v is the closest to a
10 $v_a^t[i] \leftarrow v$
11 $t_{min}[i] \leftarrow t_{arr}$
12 $t_{min}[i] \leftarrow \gamma_v$
13 **if** $v_a^t[0] \neq$ nil **then** // prioritise unallocated tasks
14 **return** $v_a^t[0]$
15 **return** $v_a^t[1]$

4.2.2 Overall Procedure of CTS

The overall procedure is described in Algorithm 4.4. The repeat-until loop is the same as CFLA2, to preserve the anytime property. Phases 1 and 2 are represented respectively by the loops at lines 5 and 16.

Phase 1 loops through all agents (line 5). Here, an agent a may either be free or reaching a task location. In the first case (line 6), if an uncompleted task v can be allocated to a (lines $7-8$), then v is flagged as allocable (line 9) and a is added to the set of agents A_v^t to which v could be allocated at time t (line 11). In the second case (line 12), a is travelling to a task v, hence its location is updated (line 13) and, if it reached v, it is set to *working on* v (lines $14-15$).

Phase 2 visits each uncompleted task v (line 16). If v is allocable (line 18) then it is allocated to the smallest coalition of agents in A_v^t (defined in Phase 1) that can complete it (lines $19-33$). In particular, at line 28, φ_v is the amount of work done by all the coalitions formed after the arrival to the location of v of the first i agents in Π_v^t (defined at line 19). After that, if there are agents working on v (line 34), its workload w_v is decreased accordingly (line 35). If w_v drops to zero or below, then v has been completed (lines $36-38$). The algorithm stops (line 40) when all the tasks have been completed, or the maximum problem time is expired, or no other tasks are allocable and uncompleted (Section 4.2.1).

Algorithm 4.4: Overall procedure of CTS (Phases 1 and 2)

Input: tasks V, task demands $(\gamma_v, w_v)_{v \in \mathrm{V}}$, agents A, locations L, travel function ρ, coalition value function u

$t \leftarrow 0$
$\Gamma' \leftarrow \emptyset$ // the solution to return (a set of coalition allocations)
$\mathrm{V}_{allocable} \leftarrow \emptyset$ // allocable tasks
repeat
  for $a \in \mathrm{A}$ **do** // Phase 1: satisfy spatial constraints
    if $a \in \mathrm{A}^t_{free}$ **then**
      $v \leftarrow$ getTaskAllocableToAgent(t, a) // Algorithm 4.3
      if $v \neq \mathit{NIL}$ **then**
        if $v \notin \mathrm{V}_{allocable}$ **then**
          $\mathrm{V}_{allocable} \leftarrow \mathrm{V}_{allocable} \cup \{v\}$
        $\mathrm{A}^t_v \leftarrow \mathrm{A}^t_v \cup \{a\}$
    else
      Update a's location
      if a reached the task v to which it was assigned **then**
        Set a's status to *working on* v
  for $v \in \mathrm{V}$ **do** // Phase 2: satisfy temporal constraints
    $\mathrm{C}^t_v \leftarrow$ all agents working on v at time t
    if $v \in \mathrm{V}_{allocable}$ **then**
      $\Pi^t_v \leftarrow$ list of all agents in A^t_v sorted by arrival time to l_v
      $\mathrm{C}^* \leftarrow \emptyset$
      for $i \leftarrow 1$ **to** $|\Pi^t_v|$ **do**
        $\mathrm{C}^* \leftarrow$ first i agents in Π^t_v
        $\lambda_i \leftarrow$ arrival time to l_v of the i-th agent in Π^t_v
        if $i + 1 \leq |\Pi^t_v|$ **then**
          $\lambda_{i+1} \leftarrow$ arrival time to l_v of the $(i+1)$-th agent in Π^t_v
        else
          $\lambda_{i+1} \leftarrow \gamma_v$
        $\varphi_v \leftarrow \varphi_v + (\lambda_{i+1} - \lambda_i) \cdot u(\mathrm{C}^* \cup \mathrm{C}^t_v, v)$ // w_v done at λ_{i+1}
        if $(\gamma_v - \lambda_{i+1}) \cdot u(\mathrm{C}^*, v) \geq w_v - \varphi_v$ **then**
          break // C^* is the minimum coalition to complete v
      $\mathrm{T}_v = \bigcup_{a \in \mathrm{C}^*} \left\{ \tau^{a \to v}_{\lambda_a} \right\}$ // λ_a is a's arrival time to l_v
      $\Gamma' \leftarrow \Gamma' \cup \Gamma(\mathrm{T}_v, t)$ // add $\Gamma(\mathrm{T}_v, t)$ (Section 3.1.2) to Γ'
      $\mathrm{V}_{allocable} \leftarrow \mathrm{V}_{allocable} \setminus \{v\}$
    if $\mathrm{C}^t_v \neq \emptyset$ **then**
      $w_v \leftarrow w_v - u(\mathrm{C}^t_v, v)$
      if $w_v \leq 0$ **then**
        Set free all agents in C^t_v
        $\mathrm{V} \leftarrow \mathrm{V} \setminus \{v\}$
  $t \leftarrow t + 1$
until $\mathrm{V} = \emptyset$ or $t > t_{max}$ or all agents are free
return Γ'

The spatial constraints (Equations 3.6 and 3.7) are satisfied by executing Algorithm 4.3 only on free agents (line 7), while the temporal constraints (Equation 3.5) are satisfied by allocating a task v to a coalition C only when C has minimum size and can complete v by deadline γ_v.

4.2.3 Analysis and Discussion

The approach of CTS transforms the CFSTP from a $1-k$ task allocation to a series of $1-1$ task allocations. In other words, instead of allocating each task to a coalition of k agents, coalitions are formed by *clustering* or grouping agents based on the closest and most urgent tasks. This is an eligibility criterion: unlike CFLA2, CTS exploits the distances between agents and tasks and the speeds of agents to reduce the time needed to define coalition allocations. Algorithm 4.3 runs in $\bar{a} = O(|V|)$ time, assuming that the operation at line 8 has constant time. In Algorithm 4.4, the time complexity of Phase 1 is $O(|A| \cdot \bar{a}) = O(|A| \cdot |V|)$, while Phase 2 runs in $O(|V| \cdot |A| \log |A|)$ because: in the worst case, $A_v^t = A$ and line 19 sorts A in $\Omega(|A| \cdot \log |A|)$ time using any sorting algorithm based on comparisons [Cormen et al. 2009]; the loop at line 21 runs in $O(|A|)$ time. Since the repeat-until loop is executed at most t_{max} times, the time complexity of Algorithm 4.4 is:

$$O\left(t_{max} \cdot |V| \cdot |A| \log |A|\right) \tag{4.2}$$

If both phases are executed in parallel, the time complexity is reduced to:

$$\Omega\left(t_{max} \cdot (|V| + |A| \log |A|)\right) \tag{4.3}$$

CTS does not have the limitations of CFLA2 (Section 4.1.3) because:

1. It can allocate more than one task per time. More formally, at each time t, if one or more tasks are allocable, CTS finds a solution with degree $1 \leq k \leq |A|$;
2. It runs in polynomial time and does not use a look-ahead technique. Thus, it is efficient and can be used in dynamic environments.

Theorem 4.1 CTS is guaranteed to find feasible coalition allocations.

Proof. We prove by induction on time t.

At $t = 0$, Phase 1 of Algorithm 4.4 selects a task v for each agent a such that v is allocable, the most urgent and closest to a (Section 4.2.1). This implies that the agent allocation $\tau_0^{a \to v}$ is legal (Section 3.1.4). Then, Phase 2 (Section 4.2.2) allocates v to a only if it exists a coalition C such that $|C|$ is minimum, $\tau_0^{C \to v}$ is feasible (Section 3.1.4) and $a \in C$.

At $t > 0$, for each agent a, there are 2 possible cases: a task v has been allocated to a at time $t' < t$, or a is free (i.e., idle). In the first case, a is either reaching or working on v (lines $12-15$

in Algorithm 4.4), hence $\tau_t^{a \to v}$ is legal and $\tau_t^{C \to v}$ is feasible, where $a \in C$. In the second case, a is either at its initial location or at the location of a task on which it finished working at time $t' < t$. Thus, as in the base case, if it exists a coalition C and a task v such that $|C|$ is minimum, $\tau_t^{C \to v}$ is feasible and $a \in C$, then v is allocated to a. □

Theorem 4.2 CTS converges to a solution, if it exists.

Proof. First we show that CTS always finds a solution, then that it terminates.

Finding a solution. From Theorem 4.1 it follows that the coalition allocations in the set $\Gamma(T_v, t)$ at line 32 of Algorithm 4.4 are feasible. That is, they satisfy all constraints (Section 3.1.4). Since $\Gamma(T_v, t)$ also defines the minimum number of agents necessary to complete task v (line 29 of Algorithm 4.4), it is a solution to v, thus it has degree $k = 1$. Therefore, if $V_{allocable} \neq \emptyset$ at some time $t \leq t_{max}$, then Γ' (line 41) has degree $k \geq 1$.

Termination. Algorithm 4.3 iterates exactly once over a finite set of uncompleted tasks, while the repeat-until loop of Algorithm 4.4 is executed at most t_{max} times. □

In finding a solution to a task v, CTS defines a coalition of minimum size among the agents that are closest to l_v. To make a comparison with an exhaustive search, this means that CTS stops at the first valid solution.

The counterexample given by Limitation 2 in Section 4.1.3 does not allow to prove the convergence of CFLA and CFLA2 in dynamic environments. Since no current algorithm that solves the CFSTP is simultaneously anytime, efficient and with convergence guarantee (Section 2.5), CTS is the first of its kind.

4.3 Comparison Tests

We implemented CFLA, CFLA2 and CTS in Java[3], and replicated the experimental setup of [Ramchurn, Polukarov et al. 2010] because we wanted to evaluate how well CFLA2 and CTS perform in settings where the look-ahead technique is highly effective. For each test configuration, we solved 100 random CFSTP instances and plotted the average and standard deviation of: percentage of completed tasks; agent travel time (Section 3.1.1); *task completion time*, or the time at which a task has no workload left; *problem completion time*, or the time at which no other tasks can be allocated.

4.3.1 Setup

Let $\mathcal{U}(l, u)$ and $\mathcal{U}^{\mathrm{I}}(l, u)$ be respectively a uniform real distribution and a uniform integer distribution with lower bound l and upper bond u. Our parameters are defined as follows:

[3] https://doi.org/10.5281/zenodo.4320671

- All agents have the same speed. Their initial locations are randomly chosen on a 50×50 grid, where the travel time of agent a between two locations is given by the Manhattan distance (i.e., the taxicab metric or ℓ_1 norm) divided by the speed of a;
- Tasks are fixed to 300, while agents range from 2 to 40, in intervals of 2 between 2 and 20 agents, and in intervals of 5 between 20 and 40 agents;
- The coalition values are defined as $u(C, v) = |C| \cdot k$, where $k \sim \mathcal{U}(1, 2)$. Hence, coalition values depend only on the number of agents involved, and all tasks have the same difficulty;
- Deadlines $\gamma_v \sim \mathcal{U}^I(5, 600)$ and workloads $w_v \sim \mathcal{U}^I(10, 50)$.

Unlike [Ramchurn, Polukarov et al. 2010], we set the number of maximum agents to 40 instead of 20, because it allows in this setup to complete all tasks in some instances. We did not perform a comparison on larger instances because of the runtime of CFLA and CFLA2: on commodity hardware, CTS takes seconds to solve instances with thousands of agents and tasks, while CFLA and CFLA2 take days. Consequently, the purpose of this section is to highlight the performance of CTS using CFLA and CFLA2 as a baseline. We evaluate the scalability of CTS in Chapter 5.

4.3.2 Results

In terms of completed tasks (Figure 4.1a), the best performing algorithm for instances with up to 18 agents is CFLA2, while the best performing algorithm for instances with at least 20 agents is CTS. CFLA is outperformed by CFLA2 in all instances except those with 2 agents, and by CTS in instances with at least 10 agents. The reason why the performance of CFLA and CFLA2 does not improve significantly starting from instances with 20 agents is that the more agents (with random initial locations) there are, the more the tasks are likely to be grouped by degree[4]. CFLA2 has a trend similar to that of CFLA because it has the same limitations, but it performs better due to its improved look-ahead technique. CTS is not the best in all instances because its average task completion time is the highest (see the discussion on Figure 4.1c below). This implies that the fewer the agents, the more the tasks may expire before they can be allocated. In our setup, 10 (resp. 20) is the number of agents starting from which this behaviour is contained enough to allow CTS to outperform CFLA (resp. CFLA2).

Regarding agent travel times (Figure 4.1b), CTS is up to 3 times more efficient than CFLA and CFLA2. This is due to Algorithm 4.3, which allocates tasks to agents also based on their proximity. CFLA2 has lower agent travel times than CFLA for the following reason. The degree computation in CFLA2 also considers how much total workload would be left (Section 4.1.2). Higher degrees correspond to lower workloads, and tasks with lower workloads are completed first. Thus, fewer tasks are grouped by degree and more are likely to be completed. This means that the average distance between task locations in a CFLA2 solution may be lower than that

[4]See Limitation 3 described in Section 4.1.3.

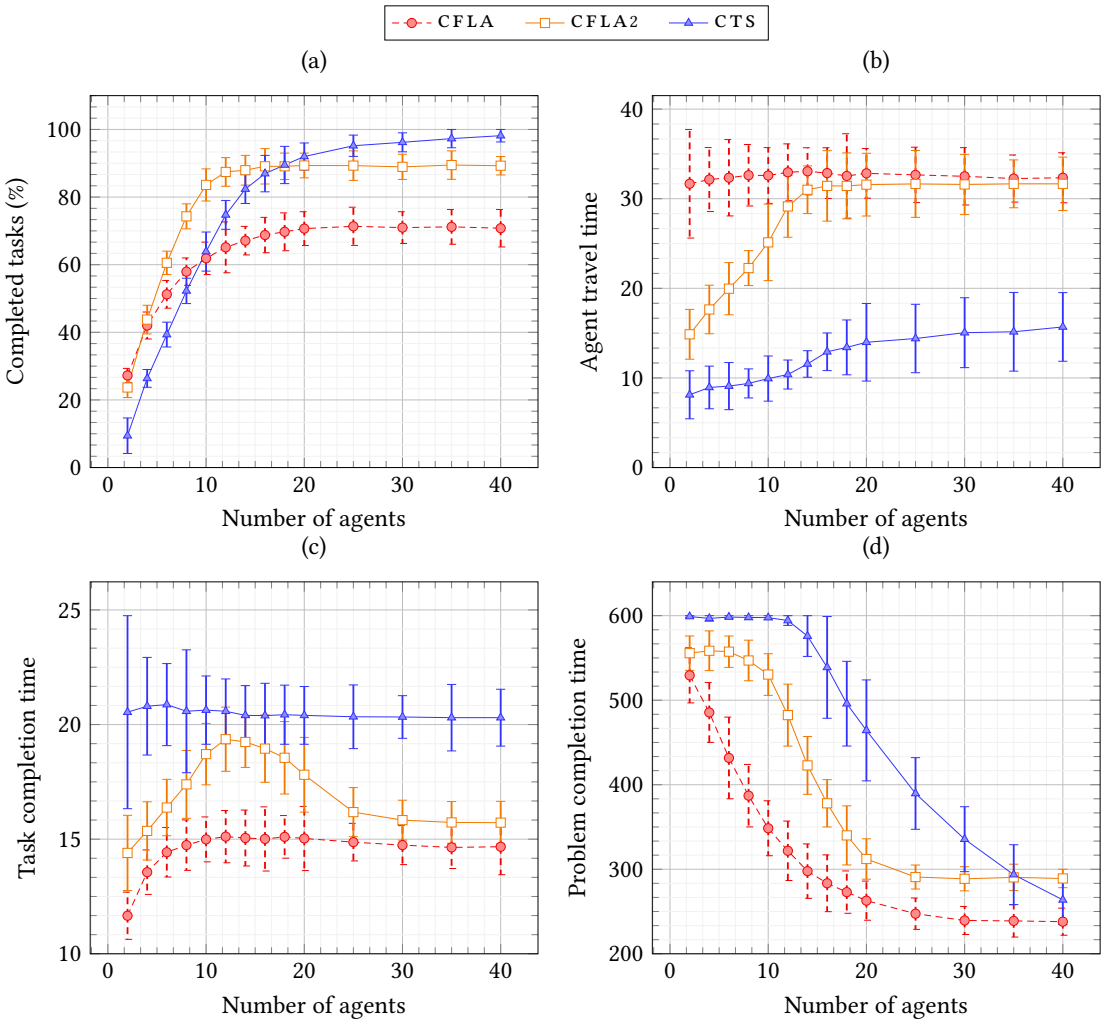


Figure 4.1 Comparison of CFLA, CFLA2 and CTS on CFSTP instances with linear coalition values. In each figure, each point is the $avg \pm std/2$, where avg is the average over 100 problems of the value indicated on the Y-axis and std is the standard deviation of avg. The tasks are fixed to 300, while the number of agents is denoted by the X-axis.

of a CFLA solution. The agent travel times increase with all algorithms. This behaviour is also reported, but not explained, by [Ramchurn, Polukarov et al. 2010]. To explain it, let us consider a toy problem with one agent a_1 and one task v. If we introduce a new agent a_2 such that $\rho(a_2, l^0_{a_2}, l_v) > \rho(a_1, l^0_{a_1}, l_v)$, then the average travel time increases. In our setup, this happens because the initial agent locations are random.

In general, task completion times (Figure 4.1c) decrease because the more agents there are, the faster tasks are completed. The completion of task v is related to the size of the coalition C to which v is allocated: the highest the completion time, the smallest the size of C, hence the highest the working time of the agents in C. Task completion times are inversely related to agent travel times. Since CTS has the shortest agent travel times and allocates tasks to the smallest coalitions, it consequently has the highest task completion times. Therefore, in CTS, agents work the highest amount of times, and the number of tasks attempted at any one time is the largest.

The problem completion times (Figure 4.1d) are in line with the task completion times (Figure 4.1c) since the faster tasks are completed, the less time is needed to solve the problem. The reason why the times of CFLA and CFLA2 do not decrease significantly from 20 agents up is linked to their performance (see the discussion on Figure 4.1a above). On the other hand, the fact that the times of CTS decrease more consistently than those of CFLA and CFLA2 indicates that CTS is the most efficient asymptotically. In other words, CTS is likely to solve large-scale problems in fewer time units than CFLA and CFLA2.

In terms of computational times, CTS is significantly faster than CFLA and CFLA2. For example, in instances with 40 agents and 300 tasks, on average[5] CTS is $45106\% \pm [2625, 32019]$ (resp. $27160\% \pm [1615, 20980]$) faster than CFLA (resp. CFLA2). The runtime improvement of CFLA2 is due to line 4 of Algorithm 4.2, which results in the look-ahead technique processing fewer tasks.

4.4 Tests with the RoboCup Rescue Simulation

In this section, we benchmark a variant of CTS against high-performance DCOP algorithms with the *RoboCup Rescue Simulation* (RCRS), one of the most important projects promoting multi-agent research on disaster response [Kitano, Tadokoro et al. 1999]. By reproducing the aftermath of an earthquake in a city, the RCRS allows verifying coordination approaches that could be enacted by first responders in such situations [Kitano and Tadokoro 2001; *RoboCup Rescue Simulator and Agent Development Framework Manual* 2021].

We conducted tests with our fork of RMASBench[6] [Kleiner et al. 2013], a benchmark platform based on the RCRS. We chose it because it allows comparisons with ready-to-use implementations of *Binary Max-Sum* (BinaryMS) [Pujol-Gonzalez, Jesus Cerquides, Farinelli, Meseguer and Juan Antonio Rodriguez-Aguilar 2015] and DSA (Page 23). We use them as a baseline because:

- Max-Sum and its variants are widely used and can obtain high quality solutions (Section 2.3). In particular, BinaryMS can produce a solution within the time limit enforced by the RCRS[7] and with the same quality as standard Max-Sum [Pujol-Gonzalez, Jesus Cerquides, Farinelli, Meseguer and Juan A. Rodriguez-Aguilar 2014; Pujol-Gonzalez, Jesus Cerquides, Farinelli, Meseguer and Juan Antonio Rodriguez-Aguilar 2015];
- Since numerous empirical evaluations have proven its efficacy in many different domains, DSA is a touchstone for testing DCOP and RCRS algorithms (Section 2.3).

Section 4.4.1 describes how CTS can be adapted for use in latest version of the RCRS, which, as of November 2021, is 1.5[8]. Section 4.4.2 reports our setup, and Section 4.4.3 shows our results.

[5] On a machine with an Intel Core i5-4690 processor (3.5 GHz, 4 threads) and 8 GB DDR3-1600 RAM.
[6] `https://doi.org/10.5281/zenodo.4320658`
[7] That is, 1 second per problem time unit.
[8] `https://github.com/roborescue/rcrs-server/releases/tag/v1.5`

Algorithm 4.5: S-CTS (executed by each agent $a \in A$)

Input: tasks V, task demands $(\gamma_v, w_v)_{v \in V}$, locations L, travel function ρ, coalition value function u

```
Initialise a's status to free
repeat
    if a's status is free then
        v ← an incomplete task with highest utility among those closest to a
        Set a's status to reaching v
    else
        Update a's location
        if a reached the task v to which it was assigned then
            Set a's status to working on v
        else if a finished working on the last allocated task then
            Set a's status to free
until V = ∅ or time out
```

4.4.1 Simplified CTS

In RCRS 1.5, deadlines and workloads are not accessible to agents. Thus, we cannot implement CTS because we can neither verify the spatial constraints in Phase 1 nor can we implement Phase 2 (Section 4.2.2). However, RMASBench allows to obtain the *utility* of a task, which is a quantitative measure that indicates the current importance of a task. Consequently, we implemented a modified Phase 1 in which agents can independently choose to work on the closest tasks with the highest utilities. Algorithm 4.5 describes this variant, which we call *Simplified CTS* (S-CTS).

The time complexity of S-CTS is $O(t_{max} \cdot |V|)$, since agents do not coordinate with each other and their choice is made in parallel (i.e., without relying on a centralised solver). Although S-CTS may seem like a major handicap, we show below that it offers a reasonable trade-off between runtime and performance.

4.4.2 Setup

All tests are based on the Paris map, one of the most used in the RoboCup competition. We kept the default setup [Pujol-Gonzalez, Jesus Cerquides, Farinelli, Meseguer and Juan Antonio Rodriguez-Aguilar 2015, Section 6.1] because, according to the authors, it maximises the performance of both BinaryMS and DSA.

In RMASBench there are police patrols and fire brigades. A police patrol can unblock roads, while a fire brigade can extinguish fires. Having 2 types of agents allows to study inter-team coordination aspects. Since this is not in our scope, we did not consider road blockades. As a result, our problems are easier and our baseline is more competitive. Figure 4.2 gives an example.

Figure 4.2 Detail of an example RCRS problem on the Paris map. The red dots are fire brigades and the blue lines are their water jets. The colour of a building indicates its status: grey means no damage; yellow to red means on fire; blue to purple means that the fire has been extinguished, and black means that the building is burnt. The darker the colour, the greater the damage. On the centre-right is a fire station, to which the fire brigades return to refill. White regions indicate irrelevant areas, such as rivers or non-flammable properties.

The RCRS is based on scenarios [*RoboCup Rescue Simulator and Agent Development Framework Manual* 2021]. A *scenario* is a class of problems, whose main parameter is the number of agents. In RMASBench there are 5, respectively with 15, 21, 27, 33 and 40 fire brigades. Other settings are:

- Agents are homogeneous, that is, they all have the same speed and water tank size;
- There are 3 ignition points, and each scenario is replicated 30 times. At each execution, a pseudo-random number generator influences the way the fires spread from ignition points to nearby buildings;
- To get a non-trivial number of fires, agents are added 25 seconds after the start;
- Each simulation runs for a maximum of 5 minutes, ending earlier if all fires have been extinguished;
- For each coalition C and task v, we have that $u(\mathrm{C}, v) = |\mathrm{C}|$, which is a special case of superadditive characteristic function [Chalkiadakis, Elkind and Wooldridge 2012, Section 2.1.2.2];
- Deadlines and workloads are randomly generated by the RCRS.

For each scenario and algorithm, we plot the average and standard deviation of:

1. Problem completion time (Section 4.3);
2. The number of buildings that burned at least once, denoted by b_{once};
3. *Score*, or the percentage of damage suffered by the city, where 100% means completely burnt. This is the main RCRS metric, defined on the total area of the city buildings and scenario-based parameters;
4. Average CPU time[9] per problem time unit.

We do not consider message-related metrics because agents do not communicate in S-CTS (Section 4.4.1).

4.4.3 Results

The more agents communicate with each other, the better they coordinate. In turn, this leads to lower completion times and numbers of burned buildings. Because there is no exchange of messages in S-CTS and BinaryMS has the highest communication overhead, they are respectively the least and the most performing in Figures 4.3a and 4.3b. Nevertheless, this does not result in a drastic drop in performance. In Figure 4.3c, in the worst-case scenario (i.e., 21 agents), on average S-CTS scores about 10% (resp. 5%) less than BinaryMS (resp. DSA). This is not trivial, given that S-CTS is a simplification and that the scenarios used are fine-tuned to maximise the performance of BinaryMS and DSA.

Regarding the average CPU time (Figure 4.3d), S-CTS is up to two orders of magnitude (resp. 1) faster than BinaryMS (resp. DSA). This is because BinaryMS has a pre-processing phase that requires exponential time, while DSA, despite having a time complexity similar to that of S-CTS (Table 2.2), has a message-passing phase as well.

In Figures 4.3a, 4.3b and 4.3c, the trends converge to 0 because the more agents there are, the less relevant the algorithm being used becomes. In other words, the greater the number of available agents, the higher the quality of solutions. We can deduce that the degree of agent communication is directly proportional to the score and inversely proportional to the CPU time. However, as we have seen, the difference in performance between communication and no communication is not necessarily significant.

4.5 Summary

We presented CFLA2, a version of CFLA with a more detailed Phase 2 and an improved Phase 3. Since we show that the time complexity of CFLA2 is quadratic in the number of tasks and exponential in the number of agents, and that the look-ahead technique cannot be used in

[9] Based on an Intel Xeon E5-2670 processor (octa-core 2.6 GHz with Hyper-Threading).

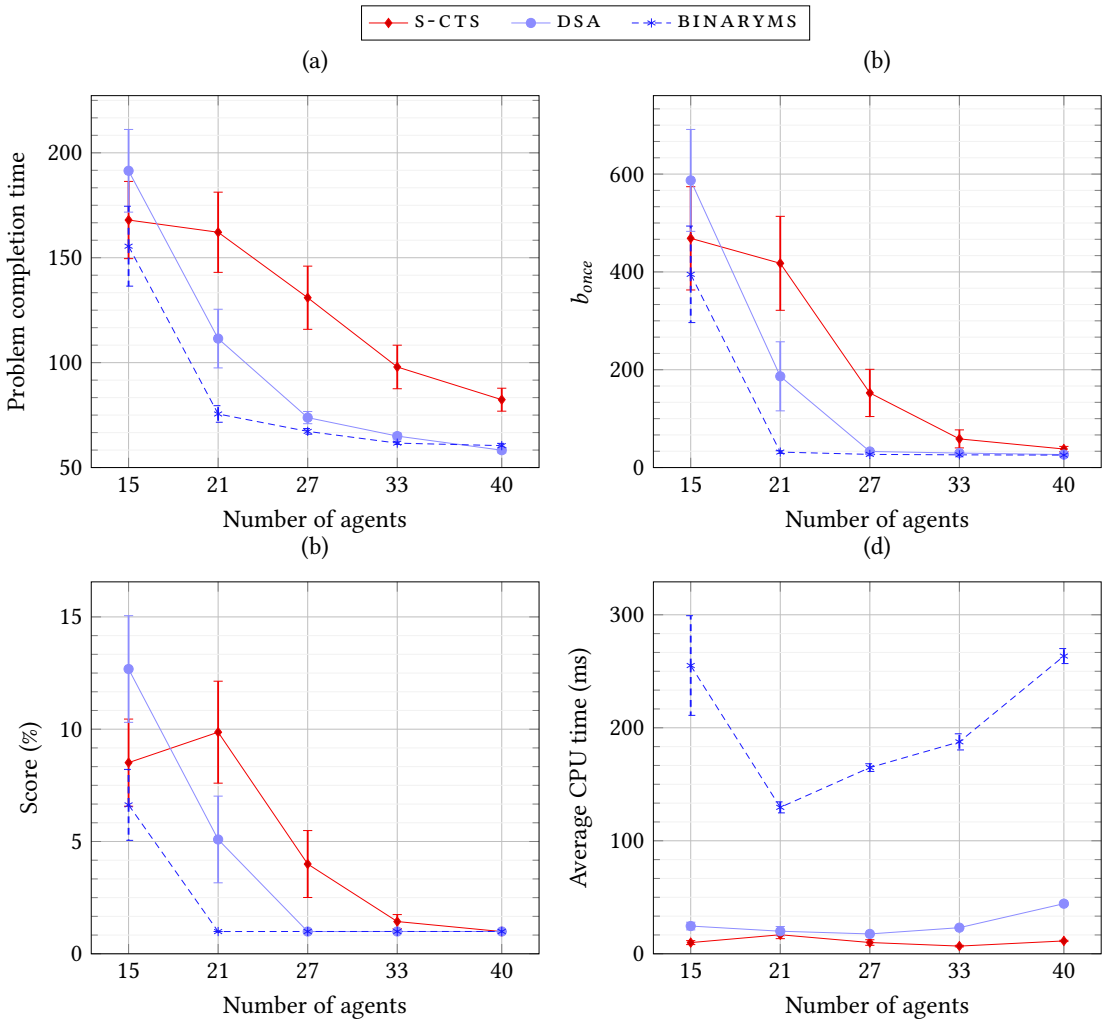


Figure 4.3 Performance of S-CTS in RMASBench using DSA and Binary Max-Sum as baselines. In each figure, the X-axis defines the number of agents in the scenario, while each point is the $avg \pm std/2$, where avg is the average over 30 simulations of the value indicated by the Y-axis, and std is the standard deviation of avg.

dynamic environments, we also presented CTS, the first anytime and efficient CFSTP algorithm with convergence guarantee. We demonstrated the superiority of CTS in settings that largely favour the look-ahead technique, and showed that a simplified but parallel variant is enough to compete with high-performance baselines in the RCRS, at a fraction of their time complexity. The next chapter extends CTS to large-scale dynamic and distributed environments, which fall within our research objectives (Section 1.3).

Chapter 5

Large-scale, Dynamic and Distributed CFSTP

After identifying the principal shortcomings in the CFSTP literature in Section 5.1, this chapter proposes a minimal mathematical formulation in Section 5.2, a distributed version of CTS in Section 5.3, and the first large-scale, dynamic and distributed CFSTP test framework in Section 5.4.

5.1 Major Gaps in the CFSTP Literature

There are 3 main issues in the CFSTP literature. First, its original mathematical programming formulation (Section 3.3) is based on 4 sets of binary variables, 1 set of integer variables and 23 types of constraints, 9 of which use the Big-M method. So many variables and constraints make implementation difficult, while the Big-M method introduces a large penalty term that, if not chosen carefully, leads to serious rounding errors and ill conditioning [Griva, Nash and Sofer 2009, Section 5.4.2].

Second, there is no algorithm that is simultaneously scalable, distributed, and able to solve the CFSTP in dynamic environments (Section 1.2). To solve this issue, we want to extend the CTS algorithm (Section 4.2), since it is anytime, has a polynomial time complexity, and can be used in dynamic environments. Its only limitation in relation to our objectives is to be centralised. In real-world domains such as disaster response, this leads to problems such as single points of failure, unsustainable computational loads, and poor performance in case of rapid changes of situation (Section 1.2). To date, only [Ramchurn, Farinelli et al. 2010] have proposed a dynamic and distributed solution to the CFSTP. They reduced it to a DynDCOP (Section 2.3.2) and solved it with FMS, a variant of Max-Sum specialised for task allocation (Page 27). However, unlike CTS, FMS is not guaranteed to converge, it is not anytime, and its runtime is exponential in the number of agents. [Pujol-Gonzalez, Jesus Cerquides, Farinelli, Meseguer and Juan Antonio

Rodriguez-Aguilar 2015] proposed BinaryMS (Section 4.4), another Max-Sum variant which, compared to FMS, lowers the runtime to polynomial and achieves the same solution quality. Nonetheless, even BinaryMS is not guaranteed to converge and not anytime. In addition, it requires a preprocessing step with exponential runtime to transform the problem constraints into binary form, which makes it not suitable for dynamic environments.

Lastly, no realistic CFSTP test framework has been proposed so far, although this shortcoming was already identified in the original article [Ramchurn, Polukarov et al. 2010, Section 8]. There are very few such frameworks even for the DynDCOP, to which the CFSTP can be reduced, as mentioned above. Indeed, although the DCOP and DynDCOP models can capture numerous real-world problems, researchers usually perform their empirical evaluations on synthetic problems or classical combinatorial problems, such as graph colouring and resource allocation [Fioretto, Pontelli and Yeoh 2018]. To the best of our knowledge, to date only the following (Dyn)DCOP works have conducted tests based on real-world data. [Maheswaran, Tambe et al. 2004] considered resource-constrained multiple-event scheduling problems occurring in office environments. [Junges and Bazzan 2008] evaluated the performance of complete DCOP algorithms in traffic light synchronisation problems. [Y. Kim, Krainin and V. Lesser 2011] developed heuristics for applying Max-Sum to problems based on the real-time sensor system NetRad. [Nelke, Okamoto and Zivan 2020; Tkach and Amador 2021] studied law enforcement problems inspired by police logs. However, none of these test frameworks is large-scale[1].

In the following sections, we address the above issues by proposing:

- A novel mathematical programming formulation of the CFSTP, based only on binary variables and 5 types of constraints, which do not use the Big-M method;
- A distributed version of the CTS algorithm that preserves its properties, namely being anytime, scalable and guaranteed to converge;
- The first large-scale, dynamic and distributed CFSTP test framework, based on real-world data published by the London Fire Brigade [London Datastore 2021a,b].

5.2 A Minimal Mathematical Program of the CFSTP

We formulate the CFSTP as a *Binary Integer Program* (BIP) [Wolsey 2020]. Based on the definitions of Sections 3.1.1 – 3.1.3, we detail our decision variables, constraints and objective function.

[1]It is worth mentioning [Leite and Enembreck 2019a], which is, to the best of our knowledge, the only study to date that evaluates incomplete DCOP algorithms in large-scale problems, although not using real-world data.

5.2.1 Decision variables

Similar to Section 3.3.1, we use the following *indicator* or binary variables:

$$\forall v \in \mathrm{V},\ \forall t \leq \gamma_v,\ \forall \mathrm{C} \subseteq \mathrm{A},\ x_{v,t,\mathrm{C}} \in \{0,1\} \tag{5.1}$$

$$\forall v \in \mathrm{V},\ y_v \in \{0,1\} \tag{5.2}$$

where: $x_{v,t,\mathrm{C}} = 1$ if coalition C works on task v at time t, and 0 otherwise; $y_v = 1$ if task v is completed, and 0 otherwise. Specifying indicator variables for individual agents is not necessary, since they can be inferred from Equation 5.1.

5.2.2 Constraints

There are 3 types of constraints: structural, temporal and spatial.

Structural Constraints At each time, at most one coalition can work on each task:

$$\forall v \in \mathrm{V},\ \forall t \leq \gamma_v,\ \sum_{\mathrm{C} \subseteq \mathrm{A}} x_{v,t,\mathrm{C}} \leq 1 \tag{5.3}$$

Equations 5.1 and 5.3 ensure that $\nexists x_{v,t,\mathrm{C}} : \mathrm{C} \nsubseteq \mathrm{A}$, and that the maximum coalition size is $|\mathrm{A}|$.

Temporal Constraints Tasks can be completed only by their deadlines, and no agent can work on a task after its completion:

$$\forall v \in \mathrm{V},\ y_v \leq 1 \tag{5.4}$$

$$\forall v \in \mathrm{V},\ \left\lceil \sum_{t \leq \gamma_v} \sum_{\mathrm{C} \subseteq \mathrm{A}} u(\mathrm{C}, v) \cdot x_{v,t,\mathrm{C}} \right\rceil = \lceil w_v \cdot y_v \rceil \tag{5.5}$$

Spatial Constraints An agent cannot work on a task before reaching its location. This identifies two cases: when an agent reaches a task from its initial location, and when an agent moves from one task location to another. The first case imposes that, for each task v, time $t \leq \gamma_v$ and coalition C, the variable $x_{v,t,\mathrm{C}}$ can be positive only if all agents in C can reach location l_v at a time $t' < t$:

$$\forall v \in \mathrm{V},\ \forall \mathrm{C} \subseteq \mathrm{A},\ \text{if } \hat{\rho} = \max_{a \in \mathrm{C}} \rho(a, l_a^0, l_v) \leq \gamma_v \text{ then } \sum_{t \leq \hat{\rho}} x_{v,t,\mathrm{C}} = 0 \tag{5.6}$$

where $\hat{\rho}$ is the maximum time at which an agent $a \in \mathrm{C}$ reaches l_v, from its initial location at time $t = 0$. Conditional constraints are usually formulated using auxiliary variables or the Big-M method [Wolsey 2020]. However, such approaches further enlarge the mathematical program or can cause numerical issues (Section 5.1). Consequently, in the preprocessing step necessary to create our BIP, we can implement Equation 5.6 simply by excluding the variables that must

be equal to 0. That is, if $\hat{\rho} \leq \gamma_v$, we only declare the following variables: $\{x_{v,t,C}\}_{t \in [\hat{\rho}+1, \gamma_v]}$. The second case requires that if an agent cannot work on two tasks consecutively, then it can work on at most one:

$$\begin{gathered} \forall v_1, v_2 \in V,\ \forall C_1, C_2 \subseteq A \text{ such that } C_1 \cap C_2 \neq \emptyset, \\ \forall t_1 \leq \gamma_{v_1},\ \forall t_2 \leq \gamma_{v_2} \text{ such that } t_1 + \max_{a \in C_1 \cap C_2} \rho(a, l_{v_1}, l_{v_2}) \geq t_2, \\ x_{v_1,t_1,C_1} + x_{v_2,t_2,C_2} \leq 1 \end{gathered} \tag{5.7}$$

Hence, coalition C_2 can work on task v_2 only if all agents in $C_1 \cap C_2$ can reach location l_{v_2} by deadline γ_{v_2}. Equation 5.7 also implies that an agent cannot work on multiple tasks at the same time.

There are no synchronisation constraints [Nunes, Manner et al. 2017]. Thus, when a task v is allocated to a coalition C, each agent $a \in C$ starts working on v as soon as it reaches its location, without waiting for the remaining agents. This means that v is completed by a temporal sequence of subcoalitions of C: $\exists S \subseteq P(C)$ such that $\forall C' \in S$, $\exists t \leq \gamma_v$, $x_{v,t,C'} = 1$, where P(C) is the power set of C.

5.2.3 Objective Function

Let $\boldsymbol{x}$ be a *solution*, that is, a value assignment to all variables, which defines the route and schedule of each agent. The objective is to find a solution that maximises the number of completed tasks:

$$\arg\max_{\boldsymbol{x}} \sum_{v \in V} y_v \text{ subject to Equations } 5.1 - 5.7 \tag{5.8}$$

Both creating all decision variables (Section 5.2.1) and finding an optimal solution exhaustively (Equation 5.8) may require to list all possible coalition allocations for each possible permutation of V, with a worst-case time complexity of:

$$O\left(|V|! \cdot 2^{|A|} \cdot t_{max}\right) \tag{5.9}$$

which is the same as the CFSTP formulation of Section 3.1.5.

Theorem 5.1 Equation 5.8 is equivalent to the original MIP of the CFSTP (Section 3.3).

Proof. Since we use the original objective function (Equation 3.38), below we show how our constraints imply the original ones (Equations 3.15 – 3.37).

Completing tasks by their deadlines and allowing agents to work only on uncompleted tasks implies that the total work done for each task v is equal to w_v if v is completed, and 0 otherwise. Hence, Equations 5.4, 5.5 ⇒ Equations 3.15, 3.16.

Equation 5.3 is equivalent to Equation 3.17.

If at most one coalition can work on each task at each time, and an agent cannot work on a task before reaching its location, then each agent can work on at most one task at each time. Consequently, Equations 5.3, 5.6 ⇒ Equation 3.21.

Equation 3.18 is not necessary because $t \leq \gamma_v$ for each $x_{v,t,\mathrm{C}}$ (Equation 5.1).

If agent a cannot work on task v before reaching its location l_v, then it can do so after finishing work on a previous task, or after reaching l_v from its initial location l_0^a. Furthermore, since the objective is to maximise the number of completed tasks, if v is allocated to a, then a changes the status of its service (from *free* to *working* and vice versa) exactly twice, and the decision variables related to a and v are equal to 1 only when a works on v, and 0 otherwise. Thus, Equations 5.6 − 5.8 ⇒ Equations 3.19, 3.20, 3.22 − 3.24.

If agent a cannot work on two tasks consecutively, then it can work on at most one. Therefore, a cannot leave and reach the same task location, it can only reach a new task location from exactly one location, and it can only leave a location to reach exactly one task location. That is, Equation 5.7 ⇒ Equations 3.25 − 3.27.

Since task v can only be completed by deadline γ_v, agent a can work on v only if it can reach its location l_v by γ_v, and the objective is to maximise the number of completed tasks, then: the coalition C of which a is a member reaches v only to complete it; C works for at least 1 unit of time on v (assuming that $w_v \geq 1$, $\forall v \in \mathrm{V}$), and each $a \in \mathrm{C}$ reaches l_v from another task location or from its initial location l_0^a. Thus, Equations 5.5 − 5.7, 5.8 ⇒ Equations 3.30 − 3.35.

Equation 5.3 is equivalent to Equation 3.36.

Since the original MIP assumes that the allocation process starts at $t = 1$ (Section 3.3.1), Equation 5.6 ⇒ Equation 3.37. If we remove this assumption, then Equation 3.37 is not necessary. □

Having significantly fewer constraints than the original MIP, our BIP can be used more effectively by exact algorithms based on branch-and-cut or branch-and-price [Vansteenwegen and Gunawan 2019, Section 3.1.1]. A trivial way to solve the CFSTP would be to implement Equation 5.8 with solvers such as CPLEX or GLPK. Although this would guarantee anytime and optimal solutions, it would also take exponential time to both create and solve our BIP (Equation 5.9). This limits this practice to offline contexts and very small problems. For example, using CPLEX 20.1 with commodity hardware, and the test setup of [Ramchurn, Polukarov et al. 2010], we can solve problems where $|\mathrm{A}| \cdot |\mathrm{V}| \leq 20$ in hours. With bigger problems, CPLEX depletes all memory (8 GB) and crashes.

Another major issue with centralised generation of optimal solutions is that, in real-time domains such as disaster response, it can be computationally not feasible or economically undesirable, especially when the problem changes frequently (Section 1.2). For these reasons, the next section presents a scalable, dynamic and distributed algorithm to solve the CFSTP.

5.3 A Scalable, Dynamic and Distributed CFSTP Algorithm

We propose a novel reduction of the CFSTP to a DynDCOP, then we show how CTS can solve the problem in this formulation. We use the DynDCOP formalism because it has proven largely capable of modelling disaster response problems (Section 2.4).

5.3.1 Reduction of the CFSTP to a DynDCOP

Using Definition 2.4, we formalise a DynDCOP as a sequence $\mathcal{D} = \{\mathcal{D}_t\}_{t \leq t_{max}}$, where each $\mathcal{D}_t = (\mathrm{A}^t, \mathrm{X}^t, \mathrm{D}^t, \mathrm{F}^t)$ is a DCOP such that $\mathrm{A}^t \subseteq \mathrm{A}$, and:

- $\mathrm{X}^t = \{\chi_1^t, \dots, \chi_k^t\}$ is a set of $k = |\mathrm{A}^t| \leq n$ variables, where χ_i^t indicates the task performed by agent $a_i^t \in \mathrm{A}^t$;
- $\mathrm{D}^t = \{\mathrm{D}_1^t, \dots, \mathrm{D}_k^t\}$ is a set of k variable domains, such that $\chi_i^t \in \mathrm{D}_i^t$. A set $d = \{d_1, \dots, d_k\}$, where $d_i \in \mathrm{D}_i^t$, is called an *assignment*. Each $d_i \in d$ is called the i-th *variable assignment* and is the value assigned to variable χ_i^t;
- $\mathrm{F}^t = \{f_1^t, \dots, f_h^t\}$ is a set of $h \leq m$ functions, where f_i^t represents the constraints on task v_i^t. In particular, each $f_i^t : \mathrm{D}_{i_1}^t \times \cdots \times \mathrm{D}_{i_{h_i}}^t \to \mathbb{R}_{\geq 0}$ assigns a non-negative real cost to each possible assignment to the variables $\mathrm{X}_{h_i}^t \subseteq \mathrm{X}^t$, where $h_i \leq h$ is the arity of f_i^t.

The objective of $\mathcal{D}$ is to find an assignment that minimises all costs:

$$\forall t \leq t_{max}, \ \arg\min_{d \in \mathrm{D}^t} \sum_{f_i^t \in \mathrm{F}^t} f_i^t(d_{i_1}, \dots, d_{i_{h_i}}) \tag{5.10}$$

It is typically assumed that if χ_i^t is in the scope of f_j^t, then agent a_i^t knows f_j^t [Fioretto, Pontelli and Yeoh 2018, Section 4.2]. To reduce the CFSTP to a DynDCOP, we specify A^t, D^t and F^t as follows. At time t, let A^t be the set of free agents (Section 3.2.2), and let $\mathrm{V}_{allocable}^t$ be the set of tasks that have not yet been completed. The domain of each variable χ_i^t is:

$$\mathrm{D}_i^t = \left\{ v \in \mathrm{V}_{allocable}^t \text{ such that } t + \rho(a_i^t, l_{a_i^t}, l_v) \leq \gamma_v \right\} \cup \{\emptyset\} \tag{5.11}$$

where $\emptyset$ means that no task is allocated to agent a_i^t. Hence, A^t satisfies the structural constraints, while D_i^t contains all tasks that at time t can be allocated to a_i^t satisfying the spatial constraints (Section 5.2.2). Let $\boldsymbol{x}_i \subseteq \boldsymbol{x}$ be a *singleton solution*, that is, a solution to task v_i (Section 5.2.3). At time t, let $\boldsymbol{x}_i^t \subseteq \boldsymbol{x}_i$ be a singleton solution corresponding to $f_i^t(d_{i_1}, \dots, d_{i_{h_i}})$, defined as follows. Each $x_{v_i,t,\mathrm{C}} \in \boldsymbol{x}_i^t$ is such that C is a subset of the agents that control the variables in the scope of f_i^t, while $x_{v_i,t,\mathrm{C}} = 1$ if $d_{i_{h_i}} = v_i$, for each h_i-th agent in C, and 0 otherwise. To satisfy the

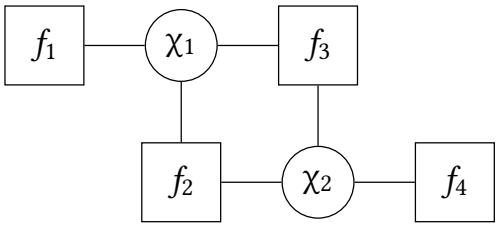


Figure 5.1 The factor graph of a DCOP with 2 agents and 4 tasks. In our formulation, a DCOP represents the state of a CFSTP at a certain time, in which circles are variables of free agents, squares are cost functions of uncompleted tasks, and each edge connects an agent to a task it can reach by its deadline.

temporal constraints (Section 5.2.2), each i-th function is defined as follows:

$$f_i^t(d_{i_1}, \dots, d_{i_{h_i}}) = \min_{\boldsymbol{x}_i^t,\, t' \leq \gamma_{v_i}} \left[\sum_{s \leq t',\, x_{v_i,s,C} \in \boldsymbol{x}_i^t} u(C, v) \right] = \lceil w_v \rceil \tag{5.12}$$

with the convention that $f_i^t(d_{i_1}, \dots, d_{i_{h_i}}) = +\infty$ if v_i cannot be completed by deadline γ_{v_i}. Hence, the solution space of $\mathcal{D}$ satisfies all CFSTP constraints, while minimising all costs implies minimising the time required to complete each task (Equations 5.10 and 5.12), which implies maximising the number of completed tasks, as required by the objective function of the CFSTP (Equation 5.8).

5.3.2 Distributed CTS

Before introducing our distributed version of CTS (Section 4.2), we present the data structure and the communication protocol on which it is based.

To represent DCOPs, we use factor graphs (Definition 2.3). As an example, Figure 5.1 shows the factor graph of the function $F(X) = f_1(\chi_1) + f_2(\chi_1, \chi_2) + f_3(\chi_1, \chi_2) + f_4(\chi_2)$. In a factor graph G, a solution is found by allowing nodes to exchange messages. Hence, to execute CTS on G, we have to define how nodes communicate and operate. Below, we present a communication protocol and algorithms for both variable and factor nodes. Based on the well-established formalism of [Yokoo, Ishida et al. 1992], the nodes communicate in the following way:

- Node i can message node j only if i knows the address of j[2]. In our context, if χ_i^t is in the scope of f_j^t, then χ_i^t knows the address of f_j^t, and vice versa;
- Each node i has a message queue Q_i, to which messages are delivered with a finite delay;
- Node i can use the function RECEIVE() to dequeue a message from Q_i, and the function SEND(j, `illoc_force`, `[args]`) to send a message to j. Node j will receive a message in the format `(sender, illoc_force, [args])`, where `sender` is the identifier of node i, `illoc_force` is its illocutionary force, and `[args]` is an optional list of arguments. An *illocutionary force* is either an information or a command [Vieira et al. 2007].

[2]For instance, the IP address of j, if the nodes are connected to the same network.

Algorithm 5.1: CTS node of variable χ_i^t

```
1  χ_i^t ← ∅ // the agent is initially free
2  d_j ← get task allocable to agent a_i^t at time t // Algorithm 4.3
3  if d_j ≠ ∅ then
4      s_i ← time at which agent a_i^t can start working on task d_j
5      f_j^t ← factor node of d_j
6      SEND(f_j^t, assignable, s_i)
7      msg ← NIL
8      while msg not received from f_j^t or not time out do
9          msg ← RECEIVE()
10     if msg = (f_j^t, allocate) then
11         χ_i^t ← d_j
```

Algorithm 5.2: CTS node of factor f_j^t

```
1  while not all neighbours sent an assignable message or not time out do
2      msg ← RECEIVE()
3  Π_{v_j}^t ← list of all assignable agents sorted by arrival time to v_j
4  C* ← minimum coalition in Π_{v_j}^t that can complete v_j by γ_{v_j} // Equation 5.12
5  for a_i^t ∈ C* do
6      SEND(χ_i^t, allocate)
7  C_{v_j}^t ← all agents working on v_j at time t
8  if C_{v_j}^t ≠ ∅ then
9      w_{v_j} ← w_{v_j} − u(C_{v_j}^t, v_j)
```

We assume that the node of each function is controlled by an agent in its scope. This agent can be chosen randomly or according to some criterion. Moreover, we assume that each agent can independently retrieve the system time (i.e., the current value of t) in constant time.

Algorithm 5.1 presents the operation of variable node χ_i^t. If there is an uncompleted task v_j^t that can be allocated to free agent a_i^t (lines $1-3$), then variable node χ_i^t communicates to factor node f_j^t the ability of a_i^t to work on v_j^t, also specifying the time at which it can reach and start working on it (lines $4-6$). After that, it waits until it gets a reply from f_j^t or a predetermined time interval expires (lines $7-9$). If it receives the approval of f_j^t, then v_j^t is allocated to a_i^t (lines $10-11$). At line 2, v_j^t is chosen such that it is the closest to a_i^t and $\gamma_{v_j^t}$ is the earliest deadline. Phase 1 is completed after that each χ_i^t executes line 6.

Algorithm 5.2 presents the operation of factor node f_j^t. The loop at lines $1-2$ is a synchronisation step that allows f_j^t to know which agents in its neighbourhood can work on v_j^t. Lines $3-6$ enacts Phase 2, while lines $7-9$ update workload w_{v_j}. Hence, our factor graphs are synchronous networks, in which the factor nodes are the synchronisers [Lynch 1996].

We call *Distributed CTS* (D-CTS) the union of Algorithms 5.1 and 5.2. Each message has size O(1), since it always contains a node address, a message flag and an integer. At time t, each

variable node χ_i^t sends at most 1 message (line 6 in Algorithm 5.1), while each factor node f_j^t sends O(|A|) messages (lines $5-6$ in Algorithm 5.2). Assuming that all tasks can be completed, the total number of messages sent is:

$$O(|A| + |V| \cdot |A|) = O(|V| \cdot |A|) \tag{5.13}$$

The runtime of Algorithm 5.1 is O(|V|), because line 2 selects a task in the neighbourhood of an agent. The runtime of Algorithm 5.2 is $O(|A| \log |A|)$, due to the sorting at line 3 [Cormen et al. 2009]. Since both algorithms are executed up to t_{max} times, the overall time complexity of D-CTS is the same as CTS (Equations 4.2 and 4.3). The advantages of D-CTS are:

1. It is anytime, since it decomposes a CFSTP into a set of subproblems (Section 4.2). This property is not trivial to guarantee in distributed systems [Zivan, Okamoto and Peled 2014], and is missing in main DCOP algorithms, such as ADOPT, DPOP, OptAPO and Max-Sum (Table 2.2);

2. It is self-stabilising (Definition 2.5), being guaranteed to converge (Theorem 4.2), and given that each agent can only work on a new task after completing the one to which it is currently assigned (Algorithm 5.1);

3. The phase-based design has 2 performance benefits. First, the algorithm is not affected by the structure of factor graphs. For instance, in a cyclic graph like the one in Figure 5.1, where the same $a > 1$ tasks can be allocated to the same $b > 1$ agents, inference-based DCOP algorithms (e.g., Max-Sum variants) in general are not guaranteed to converge, unless they are augmented with specific techniques (e.g., damping or ADVP). Second, the algorithm is robust to *disruptions*, that is, the addition or removal of factor graph nodes [Ramchurn, Farinelli et al. 2010, Section 6.2]. Disruptions are typical of dynamic environments (Section 1.2). For instance, in disaster response, tasks are removed if some victims have perished, and are added if new fires are discovered. Likewise, new agents can be added to reflect the availability of additional workforce, while existing ones are removed when they deplete their resources, or are unable to continue due to sustained damages. Unlike D-CTS, the majority of DCOP algorithms (e.g., Max-Sum and DPOP) cannot handle disruptions, unless they are properly augmented (e.g., FMS and S-DPOP);

4. Unlike most DCOP algorithms (e.g., ADOPT and DPOP), the communication overhead (i.e., the number of messages exchanged) is at most linear (Equation 5.13), and each agent does not need to maintain an information graph of all other agents;

5. Finally, performance does not depend on any tuning parameters, as is the case with other DCOP algorithms (e.g., DSA, AED and DPSA).

Since each factor node is controlled by an agent (Page 66), D-CTS is not fully distributed, but *partially centralised*. However, this is a typical assumption in DCOP algorithms that use factor graphs or pseudo-trees, as it allows to explicitly handle k-ary constraints [Fioretto, Pontelli

and Yeoh 2018, Section 4.2]. This is also the case for main algorithms such as AFB, ADOPT, DPOP and Max-Sum (Section 2.3). Partial centralisation improves local coordination and thus overall performance, but reduces privacy [Fioretto, Pontelli and Yeoh 2018, Section 4.3.1]. In cooperative coordination, and particularly in disaster response, this is generally not an issue.

Algorithms 5.1 and 5.2 resemble a *single-item auction* [Dias et al. 2006], where, for each time t and task v_j: the *bidders* are the agents in A^t that can reach v_j by γ_{v_j}; the *bid* of an agent is the time at which it can start working on v_j; the *auctioneer* is the agent controlling factor node f_j^t, which closes the auction by sending allocate messages to selected agents. However, there are two differences from a classic auction. First, v_j can be allocated to more than one agent at once. Second, agents cannot be overburdened with evaluation problems, since each bidder is interested in at most one task, and therefore each auctioneer receives as few bids as possible. In other words, the communication overhead is minimised. This advantage is due to our reduction of the CFSTP to a DynDCOP (Section 5.3.1), in which the search space contains only feasible solutions.

5.4 Empirical Evaluation in Dynamic Environments

We created a dataset[3] with 347588 tasks using open records published by the *London Fire Brigade* (LFB) over a period of 11 years. Then, we wrote a test framework in Java[4] and compared D-CTS with DSA-SDP, a state-of-the-art incomplete, synchronous and search-based DCOP algorithm (Page 23).

We adapted DSA-SDP to solve our DynDCOP formulation (Section 5.3.1), which finds a CFSTP solution by solving multiple DCOPs. Hence, being a DCOP algorithm, its performance is not penalised in our test framework. We chose it as our baseline because, similarly to D-CTS, it has a polynomial coordination overhead and is scalable (Table 2.2). We kept the parameters of [Zivan, Okamoto and Peled 2014] and ran $|V^t_{allocable}|$ iterations at each time t, since we found that, in our test framework, running more iterations can only marginally improve the solution quality, while requiring a significant increase in communication overhead and time complexity. Below, we detail our setup and discuss the results.

5.4.1 Setup

Let $\mathcal{N}$ and $\mathcal{U}$ denote the normal and uniform distribution, respectively. A test configuration consists of the following parameters:

- Since there are currently 150 identical London fire engines in operation, $|A| = 150$ for each problem. All agents have the same speed;

[3] https://doi.org/10.5281/zenodo.4728012
[4] https://doi.org/10.5281/zenodo.4764646

- $|\mathrm{V}| = |\mathrm{A}| \cdot k$, where $k \in \{1, \dots, 20\}$. Thus, problems have up to 3000 tasks;
- The demand of each task v is defined by a record dated between 1 January 2009 and 31 December 2020. More precisely, γ_v is the attendance time (in seconds) of the firefighters, and since the median attendance time in the whole dataset is about 5 minutes, we set $w_v \sim \mathcal{U}(10, 300)$ to simulate wide-ranging workloads;
- For each task-to-agent ratio $|\mathrm{V}|/|\mathrm{A}|$, the nodes of a problem are chosen in chronological order. That is, the first problem always starts with record 1, and if a problem stops at record q, then the following one will use records $q + 1$ to $q + 1 + |\mathrm{V}|$;
- The locations are latitude-longitude points, and the travel time $\rho(a, l_1, l_2)$ is given by the great-circle distance in kilometers between locations l_1 and l_2, divided by the (fixed) speed of agent a;
- In addition to task locations, L contains the locations of the 103 currently active London fire stations. In each problem, each agent starts at a fire station defined by the record of a task.

Regardless of its individual features, each agent may perform differently in different coalitions, due to the interaction with other agents. To generate coalition values, we start by taking from [Rahwan, T. Michalak and Jennings 2012, Section 4] the following well-known CSG distributions:

1. *Normally Distributed Coalition Structures* (NDCS): $u(\mathrm{C}, v) \sim \mathcal{N}(|\mathrm{C}|, \sqrt[4]{|\mathrm{C}|})$;
2. *Agent-based*: each agent a has a value $p_a \sim \mathcal{U}(0, 10)$ representing its individual performance and a value $p_a^{\mathrm{C}} \sim \mathcal{U}(0, 2 \cdot p_a)$ representing its performance in coalition C. The value of a coalition is the sum of the values of its members: $u(\mathrm{C}, v) = \sum_{a \in \mathrm{C}} p_a^{\mathrm{C}}$.

Then, we decrease each $\mu_v = u(\mathrm{C}, v)$ by values r and q, both sampled from $\mathcal{U}(\mu_v/10, \mu_v/4)$ with probability $\gamma_v/(t_{max} + 1)$ and $|\mathrm{C}|/(|\mathrm{A}| + 1)$, respectively. The perturbation r simulates real-time domains where the later γ_v is, the lower the benefit of performing v [Stankovic et al. 2013]. The perturbation q simulates situations where the more agents there are, the greater the likelihood of congestion and thus of reduced performance, as it can happen in large-scale robot swarms [Guerrero, Oliver and Valero 2017]. We call the resulting distributions UC_NDCS and UC_Agent-based, where UC means *Urgent and Congested*. NDCS assigns lower values to solutions containing fewer coalitions [Rahwan, Ramchurn et al. 2009], while Agent-based does the opposite by definition. Both distributions are neither superadditive nor subadditive [Rahwan, T. P. Michalak et al. 2015]. Hence, it is not possible to define a priori an optimal coalition for each task. We ensured consistency between the results of the algorithms by computing and storing coalition values in hash maps. More precisely, the maps were lazy-initialised and shared among all problems.

During the solution of each problem, we gradually removed agents to simulate *degradation* scenarios. The removal rate was calculated with a Poisson cumulative distribution function

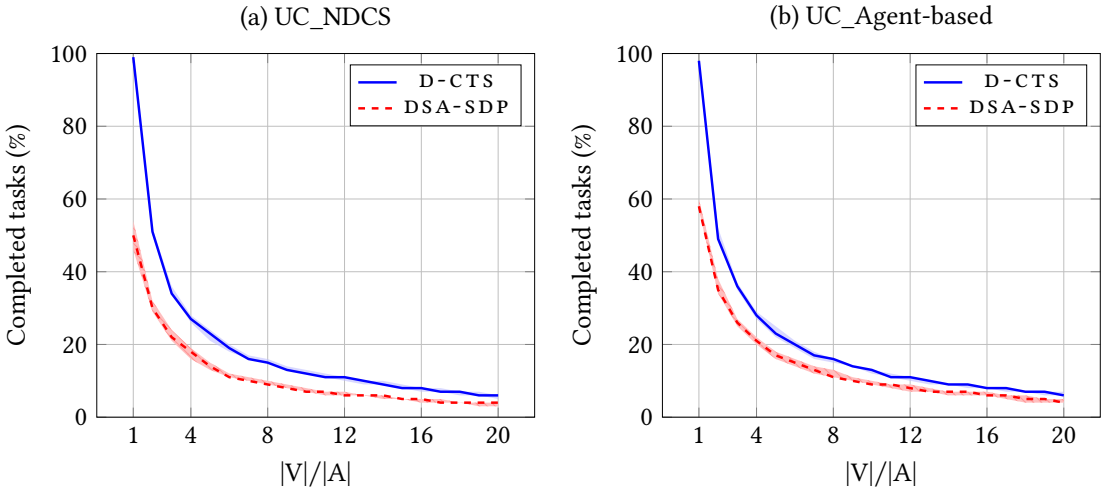


Figure 5.2 Comparison between D-CTS and DSA-SDP in our test framework. Each subfigure denotes a coalition value distribution, while each point is the median and 95% confidence interval over 100 problems of the percentage of tasks completed. The X-axis is the task-to-agent ratio.

$Pois_{\text{CDF}}(\boldsymbol{a}, \lambda)$, where $\boldsymbol{a}$ contains all firefighter arrival times in the dataset, and the rate λ is the average number of incidents per hour and per day. For each test configuration and algorithm, we solved 100 problems and measured the median and 95% confidence interval of: number of messages sent; *network load*, or the total size of messages sent; number of *Non-Concurrent Constraint Checks* (NCCCs) [Meisels 2007]; percentage of tasks completed, and CPU time[5].

5.4.2 Results

Figure 5.2 and 5.3 show our results. D-CTS completes $3.79\% \pm [42.22\%, 1.96\%]$ more tasks than DSA-SDP (Figure 5.2). For both algorithms, the performance drops rapidly as the task-to-agent ratio increases. This is due to the Urgent component in the coalition value distributions: the higher the ratio, the higher the median task completion time. Conversely, the Congested component can reduce the percentage of tasks completed more in problems with smaller task-to-agent ratios, where agents can form larger coalitions and thus increase the likelihood of congestion.

The network load of DSA-SDP is $0.59 \pm [0.41, 0.02]$ times that of D-CTS (Figure 5.3b). This is because a DSA-SDP message contains only a task address, while a D-CTS message also contains a binary flag and an integer (Section 5.3). In Java, an address requires 8 bytes, a flag requires 1 byte, and an integer requires $1 - 4$ bytes. Hence, while a DSA-SDP message always requires 8 bytes, a D-CTS message requires $10 - 13$ bytes. This is line with the results obtained. However, the situation would be reversed if we performed 1000 DSA-SDP iterations as suggested in [W. Zhang et al. 2005], since $median(\{|V^t_{allocable}|\}_{t \leq t_{max}}) \ll 1000$ in our tests.

[5]Based on an Intel Xeon E5-2670 processor (2.6 GHz, 8 threads).

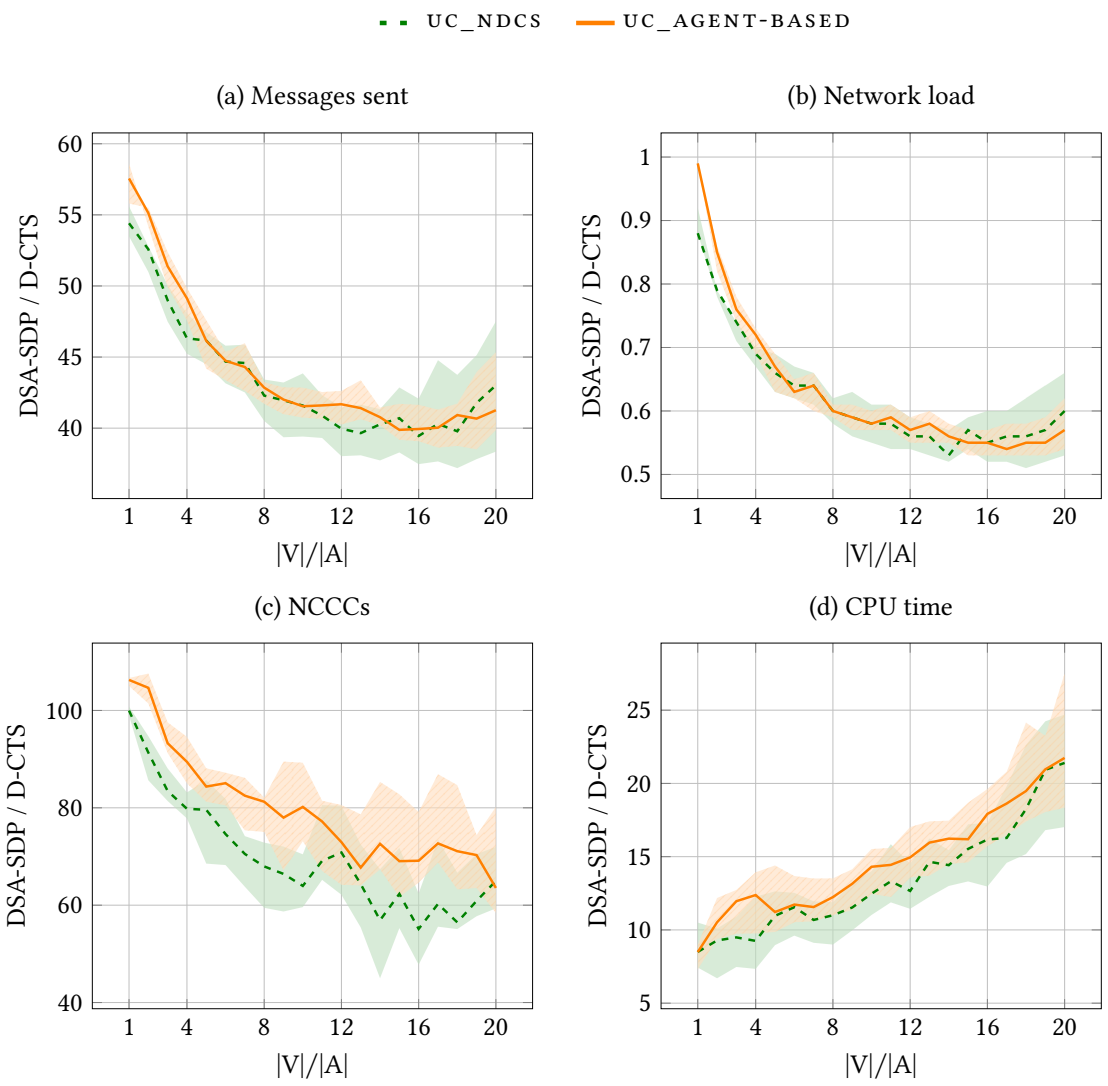


Figure 5.3 Ratio of DSA-SDP performance to D-CTS performance. Each subfigure denotes a performance metric m, while each point is the median and 95% confidence interval over 100 problems of m_a/m_b, where m_a (resp. m_b) is the value of DSA-SDP (resp. D-CTS) for m. The X-axis is the task-to-agent ratio.

The remaining metrics put DSA-SDP at a distinct disadvantage (Figure 5.3a, c, d). The overload compared to D-CTS is $41.72 \pm [12.45, 0.42]$ times more messages sent, $72.78 \pm [34.79, 27.79]$ times more NCCCs, and $13.82 \pm [4.52, 3.71]$ times more CPU time. This is explained as follows. While the number of messages sent is $O(|V| \cdot |A|)$ in D-CTS (Section 5.3), it is $O(|V| \cdot |A|^2)$ in DSA-SDP, since the agents exchange their assignments[6] [Zivan, Okamoto and Peled 2014]. In D-CTS, analysing in sequence the agents that can be assigned to each task (line 4 in Algorithm 5.2) requires $O(|V| \cdot |A|)$ NCCCs. DSA-SDP does a similar analysis, but for each message exchanged between two agents, which requires $O(|V|^2 \cdot |A|^2)$ NCCCs. Finally, the time complexity of DSA-SDP is $O(t_{max} \cdot |V| \cdot |A|^2)$, where $O(|V| \cdot |A|)$ is required by the message exchange phase at each time, and $O(|A|)$ is required by each agent to calculate the assignment costs (Equation 5.12). Hence, DSA-SDP is asymptotically slower than D-CTS (Equations 4.2 and 4.3). Overall, D-CTS took $525 \pm [281, 482]$ ms, while DSA-SDP took $6.97 \pm [5.84, 6.2]$ seconds. In accordance

[6] To align with Table 2.2, we have that $t = |V|$, and $n = l = |A|$.

with the above, the ratio of DSA-SDP performance to D-CTS performance tends to increase with regard to CPU time, and to decrease with regard to the other metrics.

In a dynamic environment, desirable features of a distributed algorithm include being robust to disruptions and minimising communication overhead (Section 5.3). The latter feature is particularly important in real-world domains such as disaster response, where agent communication can be costly (i.e., not free-comm environments, Section 2.2) or there may be operational constraints, such as low bandwidth or limited network topology (e.g., sparse robot swarms searching for shipwrecks on the seabed, or monitoring forest fires [Tarapore, Groß and Zauner 2020]). In our tests, compared to the state-of-the-art DSA-SDP, D-CTS achieves a slightly better solution quality (Figure 5.2), and is one order of magnitude more efficient in terms of communication overhead and time complexity (Figure 5.3). This affirms its effectiveness as a scalable and distributed CFSTP algorithm for dynamic environments.

5.5 Summary

We proposed a novel and minimal BIP of the CFSTP, and demonstrated its equivalence with the original MIP formulation (Section 3.3). Based on a novel reduction of the CFSTP to a DynDCOP, we also created D-CTS, a distributed version of CTS that preserves its properties (anytime, convergent, scalable) and is self-stabilising. We defined a large-scale CFSTP dataset, as well as a dynamic test framework, and empirically showed that D-CTS has slightly better performance than the state-of-the-art DSA-SDP, while having significantly lower communication overhead and time complexity. Having filled the main gaps in the literature, in order to meet all our research objectives (Section 1.3), the next and final chapter will focus on the main limitations of the CFSTP model itself.

Chapter 6

The Multi-Agent Routing and Scheduling through Coalition Formation Problem

Although many problems similar to the CFSTP have been studied to date (Section 2.5), no one can meet all our research objectives (Section 1.3). Against this background, in this final chapter we begin by presenting in Section 6.1 the *Multi-Agent Routing and Scheduling through Coalition formation problem* (MARSC), a generalisation of the CFSTP and the TOPTW that can be used in real-time domains. Then, we define in Section 6.2 the first anytime, exact and parallel MARSC algorithm. Since no exact algorithm has so far been proposed for the CFSTP, it is consequently the first of its kind for this problem as well. Finally, in Section 6.3, using extended versions of the test frameworks of Chapters 4 and 5, we evaluate our algorithm on synthetic small-scale problems, and an approximate variant on large-scale realistic problems.

6.1 Problem Formulation

As in Section 5.2, we use a BIP formulation. Since the MARSC generalises the CFSTP (Theorem 6.1), we recall and extend the basic definitions and constraints of Sections 3.1.1 − 3.1.3 and 5.2. Specifically, we add multiple possible locations per task, benefits, time windows, precedences, and define a more general objective function. We also show how to extend the work of Section 5.3.1 to reduce the MARSC to a DynDCOP.

6.1.1 Basic Definitions

Let $\mathrm{V} = \{v_1, \dots, v_m\}$ be a set of m tasks and $\mathrm{A} = \{a_1, \dots, a_n\}$ be a set of n agents. Let L be the finite set of all possible task and agent locations. Time is denoted by $t \in \mathbb{N}$, starting at $t = 0$, and

agents travel or perform tasks with a base time unit of 1. The time units needed by an agent to travel from one location to another are given by the function $\rho : A \times L \times L \to \mathbb{N}$. Having A in the domain of ρ allows to characterise different agent features (e.g., speed or type). Let $l_a^t \in L$ be the location of agent a at time t, where l_a^0 is the initial location of a and is known a priori.

Task Demand Each task v has a *demand* $D_v = (L_v, w_v, \phi_v, \alpha_v, \beta_v, \gamma_v)$, where:

- $L_v \subseteq L$ is the *set of possible locations of* v;
- $w_v \in \mathbb{R}_{\geq 0}$ is the *workload* of v, or the amount of work required to complete v;
- $\phi_v \in \mathbb{R}_{\geq 0}$ is the *benefit* of v, or the weight associated with the completion of v;
- $[\alpha_v, \beta_v, \gamma_v]$ is the *time window* of v, such that $\alpha_v \in \mathbb{N}$ is the *earliest time* of v, or the time starting from which agents can work on v, $\beta_v \in \mathbb{N}$ is the *soft latest time* of v, or the time until which agents can work on v without incurring in a penalty, and $\gamma_v \in \mathbb{N}$ is the *hard latest time* of v, or the time until which agents can work on v incurring in a penalty. No agent can work on v after γ_v.

We assume that $\alpha_v \leq \beta_v \leq \gamma_v$, $\forall v \in V$, and call $t_{max} = \max_{v \in V} \gamma_v$ the *maximum problem time*.

Task Order Let $Prec \subseteq V \times V$ be such that if $(v_1, v_2) \in Prec$ then v_1 must be completed before v_2. Each $(v_1, v_2) \in Prec$ is called a *precedence* and can be graphically denoted with $v_1 \prec v_2$. As typically done in MRTA [Nunes, Manner et al. 2017], we assume that $G = (V, Prec)$ is a finite, directed and acyclic graph. Moreover, without loss of generality, we assume that $Prec$ does not contain relations which can be inferred transitively, that is:

$$(v_1, v_2) \in Prec \wedge (v_2, v_3) \in Prec \Rightarrow (v_1, v_3) \notin Prec$$

Coalition and Coalition Value A subset of agents $C \subseteq A$ is called a *coalition*. For each coalition, task and location there is a *coalition value*, given by the function $u : P(A) \times V \times L \to \mathbb{R}_{\geq 0}$, where $P(A)$ is the power set of A. The value of $u(C, v, l)$ is the amount of work that coalition C does on task v at location $l \in L_v$ in one time unit. In other words, when C performs v in l, $u(C, v, l)$ expresses how well the agents in C work together, and the workload w_v decreases by $u(C, v, l)$ at each time.

6.1.2 Decision Variables

We use the following indicator variables:

$$\forall v \in V,\ \forall l \in L_v,\ \forall t \in [\alpha_v, \gamma_v],\ \forall C \subseteq A,\ x_{v,l,t,C} \in \{0, 1\} \tag{6.1}$$

$$\forall v \in V,\ \forall l \in L_v,\ y_{v,l} \in \{0, 1\} \tag{6.2}$$

where: $x_{v,l,t,\mathrm{C}} = 1$ if task v in location l and at time t is performed by coalition C, and 0 otherwise; $y_{v,l} = 1$ if task v is completed in location l, and 0 otherwise.

6.1.3 Constraints

There are 4 types of constraints: structural, temporal, spatial and ordering.

Structural Constraints Each task can be performed by at most one coalition at each time, and only within its time window:

$$\forall v \in \mathrm{V},\ \forall l \in \mathrm{L}_v,\ \forall t \in [\alpha_v, \gamma_v],\ \sum_{\mathrm{C} \subseteq \mathrm{A}} x_{v,l,t,\mathrm{C}} \leq 1 \tag{6.3}$$

Temporal Constraints Each task can be performed in at most one location, and no agent can work on it after its workload has been completed:

$$\forall v \in \mathrm{V},\ \sum_{l \in \mathrm{L}_v} y_{v,l} \leq 1 \tag{6.4}$$

$$\forall v \in \mathrm{V},\ \forall l \in \mathrm{L}_v,\ \left\lfloor \sum_{t \in [\alpha_v, \gamma_v]} \sum_{\mathrm{C} \subseteq \mathrm{A}} u(\mathrm{C}, v, l) \cdot x_{v,l,t,\mathrm{C}} \right\rfloor = \left\lceil w_v \cdot y_{v,l} \right\rceil \tag{6.5}$$

Spatial Constraints An agent cannot work on a task before reaching one of its possible locations. This identifies two cases: when an agent reaches a task location from its initial location, and when an agent moves from one task location to another. The first case imposes that, for each task v, location $l \in \mathrm{L}_v$ and coalition C, the decision variable $x_{v,l,t,\mathrm{C}}$ can be positive only if all agents in C can reach l at time $t' < t$:

$$\forall v \in \mathrm{V},\ \forall l \in \mathrm{L}_v,\ \forall \mathrm{C} \subseteq \mathrm{A},\ \text{if } \hat{\rho} = \max_{a \in \mathrm{C}} \rho(a, l_a^0, l) \geq \alpha_v \text{ then } \sum_{t \in [\alpha_v, \hat{\rho}]} x_{v,l,t,\mathrm{C}} = 0 \tag{6.6}$$

The value of $\hat{\rho}$ is the maximum time at which an agent $a \in \mathrm{C}$ reaches l, from its initial location at time $t = 0$. Conditional constraints are usually formulated using auxiliary variables or the Big-M method. However, such approaches further enlarge the mathematical program or easily cause numerical issues (Section 5.1). Consequently, in the preprocessing step necessary to create our BIP, we can implement Equation 6.6 simply by excluding the variables that must be equal to 0. The second case requires that if an agent cannot perform two tasks consecutively, then it can perform at most one:

$$\begin{gathered} \forall \mathrm{C}_1, \mathrm{C}_2 \subseteq \mathrm{A} : \mathrm{C}_1 \cap \mathrm{C}_2 \neq \emptyset,\ \forall v_1, v_2 \in \mathrm{V},\ \forall l_1 \in \mathrm{L}_{v_1},\ \forall l_2 \in \mathrm{L}_{v_2}, \\ \forall t_1 \in [\alpha_1, \gamma_1],\ \forall t_2 \in [\alpha_2, \gamma_2] : t_1 + \max_{a \in \mathrm{C}_1 \cap \mathrm{C}_2} \rho(a, l_1, l_2) \geq t_2, \\ x_{v_1,l_1,t_1,\mathrm{C}_1} + x_{v_2,l_2,t_2,\mathrm{C}_2} \leq 1 \end{gathered} \tag{6.7}$$

Hence, coalition C_2 can perform task v_2 only if all agents in $C_1 \cap C_2$ can reach location l_2 by the hard latest time γ_2. Equation 6.7 also implies that two tasks cannot be performed by the same agent at the same time. Consequently, coalitions that exist in different locations at the same time are disjoint. There are no synchronisation constraints [Nunes, Manner et al. 2017]. Thus, when a task v in location l is allocated to a coalition C, each agent $a \in C$ starts working on v as soon as it reaches l, without waiting for the remaining agents. That is, v is completed by a temporal sequence of subcoalitions of C: $\exists S \subseteq P(C)$ such that $\forall C' \in S$, $\exists t \in [\alpha_v, \gamma_v]$, $x_{v,l,t,C'} = 1$, where P(C) is the power set of C.

Ordering Constraints If $v_1 \prec v_2$ then v_2 can only be performed after v_1:

$$\forall v_1, v_2 \in V, \text{ if } v_1 \prec v_2 \text{ then } \sum_{l_1 \in L_{v_1}} y_{v_1,l_1} \geq \sum_{l_2 \in L_{v_2}} y_{v_2,l_2} \tag{6.8}$$

6.1.4 Objective Function

Let $\boldsymbol{x}$ be a *solution*, that is, a value assignment to all decision variables, which defines the route and schedule of each agent. For each task v and time t, let:

$$\psi_{v,t} = \begin{cases} 1, & \text{if } t \leq \beta_v \\ 1 - \dfrac{t - \beta_v}{\gamma_v - \beta_v + 1}, & \text{otherwise} \end{cases} \tag{6.9}$$

be the *penalty* of performing v at t, that is, a positive weight that decreases linearly after t passes β_v. Let:

$$score(\boldsymbol{x}) = \sum_{v \in V} \sum_{l \in L_v} \sum_{t \in [\alpha_v, \gamma_v]} \sum_{C \subseteq A} \phi_v \cdot \psi_{v,t} \cdot x_{v,l,t,C} \tag{6.10}$$

be the *score* of $\boldsymbol{x}$. The objective of the MARSC is to find a solution that has maximum score and satisfies all constraints:

$$\arg\max_{\boldsymbol{x}} score(\boldsymbol{x}) \text{ subject to Equations } 6.1 - 6.8 \tag{6.11}$$

Hence, Equation 6.11 maximises the total benefit while minimising the total time penalty (i.e., the number of tasks that are completed after their soft latest times). This involves maximising task completion times and minimising coalition sizes. Consequently, the number of tasks performed at any one time is the largest, and the most beneficial tasks are prioritised. Ours is a combination of two objective functions commonly used in MRTA problems [Nunes, Manner et al. 2017, Section 5]. Creating all decision variables (Equation 6.1) may require to list all $\mathcal{L}$-tuples over P(A), where $\mathcal{L} = |V| \cdot |L| \cdot t_{max}$. This implies a worst-case time and space complexity of:

$$O\left(\left(2^{|A|}\right)^{\mathcal{L}}\right) = O\left(2^{|A| \cdot |V| \cdot |L| \cdot t_{max}}\right) \tag{6.12}$$

In addition, searching for an optimal solution exhaustively (Equation 6.11) may require to list all possible coalition allocations for each possible permutation of V, with a worst-case time complexity of:

$$\mathrm{O}\left(|\mathrm{V}|! \cdot |\mathrm{L}| \cdot 2^{|\mathrm{A}|} \cdot t_{max}\right) \tag{6.13}$$

Theorem 6.1 The MARSC generalises the CFSTP.

Proof. We refer to the BIP formulation of the CFSTP given in Section 5.2.

The CFSTP is a MARSC where tasks have exactly one location, benefits are homogeneous, there are neither earliest nor soft latest times, and tasks can be completed in any order. The structural, temporal and spatial constraints are identical. Given a MARSC solution $\boldsymbol{x}$ and a task v, if $x_{v,l,t,\mathrm{C}} = 1$, for some $l \in \mathrm{L}_v$, $t \in [\alpha_v, \gamma_v]$ and $\mathrm{C} \in \mathrm{P(A)}$, then $y_{v,l} = 1$. Thus, in the presence of the above-mentioned simplifications, Equation 6.11 maximises the number of completed tasks, as required by the objective function of the CFSTP (Equation 5.8). □

Theorem 6.2 The MARSC generalises the TOPTW.

Proof. Based on [Vansteenwegen and Gunawan 2019, Section 3.3], we first describe the TOPTW using our terminology (Section 6.1.1), then we show that it is a special case of the MARSC.

The TOPTW considers a finite set of tasks, each with a non-negative benefit. Each task has exactly one location and no soft latest time: $|\mathrm{L}_v| = 1 \wedge \beta_v = \gamma_v, \forall v \in \mathrm{V}$. When a task is completed within its time window, its benefit is added to the total benefit. Between each pair of tasks there is a fixed travel time. There are an initial task and a final task, both with a benefit of 0, an earliest time of 0 and a soft latest time of t_{max}. Each route must begin at the location of the initial task and end at the location of the final task. The objective is to determine n routes, one for each agent, that maximise the total benefit.

The travel time between tasks v_i and v_j also includes the *service time* at v_j, that is, the time taken by any coalition to complete v_j [Vansteenwegen and Gunawan 2019, Section 2.2]. Hence, we set $w_v = 1, \forall v \in \mathrm{V}$, to ensure that each task is completed in at most 1 unit of time. Since travel times depend only on task locations, we exclude A from the domain of $\rho(\cdot)$. Coalitions are singleton, that is, each C is such that $|\mathrm{C}| = 1$. The coalition value function $u(\cdot)$ always returns 1.

Because each task has exactly one location and coalitions are singleton, we remove the subscript l from the decision variables and simply use a instead of C to indicate the coalition that consists of agent a. Hence: $x_{v,t,a} = 1$ if agent a works on task v at time t, and 0 otherwise; $y_v = 1$ if task v is completed, and 0 otherwise. Let v_{start} and v_{end} denote the initial and final tasks, respectively. The structural constraints (Equation 6.3) become:

$$\forall v \in \mathrm{V} \setminus \{v_{start}, v_{end}\},\ \forall t \in [\alpha_v, \beta_v],\ \sum_{a \in \mathrm{A}} x_{v,t,a} \leq 1 \tag{6.14}$$

Since both coalition values and workloads are unitary, and each task does not have multiple locations, Equation 6.14 already ensures that a task is completed within its time window by at most one agent. Hence, no temporal constraints (Equations 6.4 and 6.5) are required.

Let l_{start} denote the location of v_{start}, and let l_v denote the location of any other task v. The initial location of each agent is l_{start}, and the spatial constraints (Equations 6.6 and 6.7) become:

$$\forall v \in \mathrm{V},\ \forall a \in \mathrm{A},\ \text{if } \lambda = \rho(l_{start}, l_v) \geq \alpha_v \text{ then } \sum_{t \in [\alpha_v, \lambda]} x_{v,t,a} = 0 \tag{6.15}$$

$$\begin{gathered} \forall v_1, v_2 \in \mathrm{V},\ \forall a \in \mathrm{A},\ \forall t_1 \in [\alpha_1, \beta_1],\ \forall t_2 \in [\alpha_2, \beta_2] \ :\ t_1 + \rho(l_{v_1}, l_{v_2}) \geq t_2, \\ x_{v_1,t_1,a} + x_{v_2,t_2,a} \leq 1 \end{gathered} \tag{6.16}$$

To ensure that each agent schedule starts with v_{start} and ends with v_{end}, we define the set of precedences as follows:

$$Prec = \{(v_{start}, v), (v, v_{end}) \ :\ v \in \mathrm{V} \setminus \{v_{start}, v_{end}\}\}$$

The ordering constraints (Equation 6.8) become:

$$\forall v_1, v_2 \in \mathrm{V},\ \text{if } v_1 \prec v_2 \text{ then } y_{v_1} \geq y_{v_2} \tag{6.17}$$

The absence of soft latest times means that there can be no time penalties: $\psi_{v,t} = 1$, $\forall v \in \mathrm{V}$, $\forall t \leq t_{max}$ (Equation 6.9). Hence, the objective function (Equation 6.11) is simplified as follows:

$$\begin{gathered} \arg\max_{\boldsymbol{x}} \sum_{v \in \mathrm{V} \setminus \{v_{start}, v_{end}\}} \sum_{t \in [\alpha_v, \beta_v]} \sum_{a \in \mathrm{A}} \phi_v \cdot x_{v,t,a} \\ \text{subject to Equations } 6.14 - 6.17 \end{gathered} \tag{6.18}$$

□

Both the CFSTP and the TOPTW are NP-hard [Ramchurn, Polukarov et al. 2010; Vansteenwegen and Gunawan 2019], thus the MARSC is NP-hard as well.

6.1.5 Reduction of the MARSC to a DynDCOP

Given that the MARSC generalises the CFSTP (Theorem 6.1), to reduce it to a DynDCOP, there are two modifications to be made to the work of Section 5.3.1.

Let $\mathrm{V}^t_{completed}$ denote the set of tasks that have been completed at time t. The first modification is to include ordering constraints and the possibility of having multiple possible locations per task by redefining Equation 5.11 as follows:

$$\begin{gathered} \mathrm{D}^t_i = \Big\{ v \in \mathrm{V}^t_{allocable} \,|\, \big(\exists v_2 \in \mathrm{V} \ :\ v_2 \prec v \Rightarrow v_2 \in \mathrm{V}^t_{completed}\big) \ \wedge \\ \exists l \in \mathrm{L}_v \ :\ t + \rho(a^t_i, l_{a^t_i}, l) \leq \gamma_v \Big\} \cup \{\emptyset\} \end{gathered} \tag{6.19}$$

Recall that, in propositional logic, $A \Rightarrow B \equiv \neg A \vee B \equiv$ if A then B. The second change is to include time windows, benefits and penalties by redefining Equation 5.12 as follows:

$$f_i^t(d_{i_1}, \dots, d_{i_{h_i}}) = \min_{\boldsymbol{x}_i^t} -score(\boldsymbol{x}_i^t) \tag{6.20}$$

where each $x_{v_i,l,t,C} \in \boldsymbol{x}_i^t$ is such that C is a subset of the agents that control the variables in the scope of f_i^t, while $x_{v_i,l,t,C} = 1$ if $d_{i_{h_i}} = v_i$, for each h_i-th agent in C and for some $l \in L_{v_i}$, and 0 otherwise. Being an additive function (Equation 6.10), the solution score fits naturally with the characterisation of $f_i^t(\cdot)$. It is multiplied by -1 in Equation 6.20 because we formulate the DynDCOP as a minimisation problem (Equation 5.10).

6.2 An Anytime, Exact and Parallel MARSC Algorithm

A trivial way to solve the MARSC would be to implement Equation 6.11 with solvers such as CPLEX or GLPK. Although this would guarantee anytime and optimal solutions, it would also take exponential time and space to create the BIP (Equation 6.12), in addition to the time to solve it (Equation 6.13). This limits this practice to offline contexts and very small problems. For example, using CPLEX 20.1 with our HPC cluster[1], we can solve within hours problems with superadditive coalition values, uniformly distributed workloads and time windows, locations based on the taxicab metric, and $dim = |A| \cdot |V| \cdot |L| \leq 30$. With greater dim values, CPLEX depletes all memory (187.5 GB) and crashes. For this reason, this section presents the *Anytime, exact and parallel Node Traversal* (ANT) algorithm, which provides the sharp lower bound on the time and space complexity required to solve the MARSC optimally. We begin by explaining its procedures, then we analyse its theoretical properties and computational complexity.

6.2.1 Procedures

Given a task v, a location $l \in L_v$ and a set of agents $A' \subseteq A$, Algorithm 6.1 defines a coalition $C^* \subseteq P(A')$ such that $|C^*|$ is minimum and each agent $a \in C^*$ reaches l not only by γ_v (line 1), but also in the shortest possible time (line 2), which satisfies the spatial constraints. The status of each agent (line 1) is set by Algorithm 6.2. When the condition at line 12 holds, C^* satisfies the temporal constraints. Line 13 returns a singleton solution $\boldsymbol{x}_v \subseteq \boldsymbol{x}$ (Section 5.3.1). In particular, $\boldsymbol{x}_v$ defines the temporal sequence of subcoalitions of C^* that are formed as the agents reach l, thus satisfying the structural constraints. That is, for each $a \in C^*$, $\boldsymbol{x}_v$ defines the time interval i during which a will work on v at l, such that i does not violate any constraints, and $score(\boldsymbol{x}_v)$ is maximum; a is considered part of C^* from the moment $\boldsymbol{x}_v$ is created until the end of i.

Let $\sigma(V)$ denote the set of all permutations of V. Algorithm 6.2 takes as input a task permutation $V^\alpha \subseteq \sigma(V)$ and a set of agents $A' \subseteq A$. It defines a solution $\boldsymbol{x}^\alpha$ by finding singleton solutions

[1]The Iridis 5 Compute Cluster at the University of Southampton.

Algorithm 6.1: getSingletonSolution

Input: task v, location $l \in \mathrm{L}_v$, agents A'

1 $\Pi_v \leftarrow$ array of all agents in A' that, from their current status, can reach l by γ_v
2 Sort Π_v by arrival time to l
3 $\varphi_v \leftarrow 0$ `// total workload done on` v
4 **for** $i \leftarrow 1; i \leq |\Pi_v|; i \leftarrow i + 1$ **do**
5 $\quad \mathrm{C}^* \leftarrow$ first i agents in Π_v
6 $\quad \lambda_i \leftarrow$ arrival time to l of the i-th agent in Π_v
7 $\quad$ **if** $i + 1 \leq |\Pi_v|$ **then**
8 $\quad\quad \lambda_{i+1} \leftarrow$ arrival time to l of the $(i+1)$-th agent in Π_v
9 $\quad$ **else** `// the` i`-th agent is the last`
10 $\quad\quad \lambda_{i+1} \leftarrow \gamma_v$ `// last feasible working time`
11 $\quad \varphi_v \leftarrow \varphi_v + (\lambda_{i+1} - \lambda_i) \cdot u(\mathrm{C}^*, v, l)$ `// workload done at time` λ_{i+1}
12 $\quad$ **if** $(\gamma_v - \lambda_{i+1}) \cdot u(\mathrm{C}^*, v, l) \geq w_v - \varphi_v$ **then**
13 $\quad\quad$ **return** $\boldsymbol{x}_v$ `// using` C^*`,` v`, and` l
14 **return** NIL

Algorithm 6.2: getSolutionForSchedule

Input: incumbent solution $\boldsymbol{x}$, task permutation V^α, agents A'

1 $\boldsymbol{x}^\alpha \leftarrow$ empty vector
2 **for** $v \in \mathrm{V}^\alpha$ **do**
3 $\quad \boldsymbol{x}^\alpha_v \leftarrow$ NIL
4 $\quad$ **for** $l \in \mathrm{L}_v$ **do**
5 $\quad\quad \boldsymbol{x}^\alpha_{v,l} \leftarrow$ getSingletonSolution(v, l, A')
6 $\quad\quad \boldsymbol{x}^\alpha_v \leftarrow \arg\max_{\boldsymbol{x}' \in \{\boldsymbol{x}^\alpha_{v,l}, \boldsymbol{x}^\alpha_v\}} score(\boldsymbol{x}')$
7 $\quad$ **if** $\boldsymbol{x}^\alpha_v \neq$ NIL **then**
8 $\quad\quad \forall a \in \mathrm{A}'$, update a.status based on $\boldsymbol{x}^\alpha_v$
9 **return** $\arg\max_{\boldsymbol{x}' \in \{\boldsymbol{x}^\alpha, \boldsymbol{x}\}} score(\boldsymbol{x}')$ `// access to` $\boldsymbol{x}$ `is synchronised`

following the order of V^α. For each $v \in \mathrm{V}^\alpha$, the loop at line 4 finds a location $l \in \mathrm{L}_v$ that maximises $score(\boldsymbol{x}^\alpha_v)$, with the convention that $score(\text{NIL}) = 0$. Lines 7 and 8 ensure that $\boldsymbol{x}^\alpha$ satisfies the structural constraints by saving, after finding each singleton solution and for each agent $a \in \mathrm{A}'$, the last time a was working, and at which location (i.e., its status). At line 9, $\boldsymbol{x}$ is maximised synchronously to allow execution in concurrent environments.

Algorithm 6.3 describes the overall procedure. Lines 2 and 3 initialise the incumbent solution using the *Earliest Deadline First* (EDF) technique, which is typically used in real-time systems [Stankovic et al. 2013]. Lines $4-6$ launch a thread with an instance of Algorithm 6.2 for each *schedule* or permutation of V that satisfies the ordering constraints. At lines 3 and 6, A is cloned to avoid interferences between threads. The stopping criterion at line 7 works as follows. Let k be the number of tasks performed in the solution found at line 6, let S be the set of all schedules investigated so far, and let $\tilde{\mathrm{X}}$ be the set of all solutions found so far. If for each permutation of k tasks p there is a schedule in S that starts with p, and $\tilde{\mathrm{X}}$ does not contain a solution involving

Algorithm 6.3: ANT

Input: tasks V, demands $\{D_v\}_{v \in V}$, order $Prec$, agents A

```
x ← empty vector
Sort V by earliest time, while satisfying the ordering constraints // initial schedule
x ← getSolutionForSchedule(x, V, clone(A)) // initial incumbent solution
for V^α ∈ σ(V) do // remaining schedules
    if V^α satisfies the ordering constraints then
        x ← getSolutionForSchedule(x, V^α, clone(A))
        if ∃k ≤ |V| : ∄ solutions for k tasks in x then
            break
return x
```

k tasks, then there can be no solution for $k' \geq k$ tasks. Since S consequently also contains schedules starting with each permutation of $k'' < k$ tasks, the search can safely end.

The reason why it is sufficient that, for each k-permutation of tasks p, S contains a schedule s *starting* with p, is that s has maximum score for subschedule p. In other words, due to Equation 6.10, any schedule $s' \neq s$ containing p (in any position) is such that $score(s') \leq score(s)$. Figure 6.1 shows an example of the execution of Algorithm 6.3.

6.2.2 Theoretical Properties

Algorithm 6.1 is based on the following lemma.

Lemma 6.3 $\forall a \in A, \forall v \in V, \forall t \in [\alpha_v, \gamma_v]$, if $\rho(a, l_a^0, l_v) > t$, then $\nexists v_2 \in V \setminus \{v\}$ such that:

$$\exists l_{v_2} \in L_{v_2},\ \exists t_2 \in [\alpha_{v_2}, \gamma_{v_2}] \ :\ t_2 + \rho(a, l_{v_2}, l_v) \leq t \tag{6.21}$$

Proof. If agent a cannot reach location l_v from its initial location l_a^0 by time t, then it cannot reach l_v by t even if it would depart from any other location at time $t' < t$, because each possible route of a always starts at l_a^0. □

Hence, if $\rho(a, l_a^0, l_v) > t$, then we can safely exclude each decision variable $x_{v,l,t,C}$ such that $a \in C$. This means that only agents that can reach l_v by γ_v are able to form feasible coalitions, and only coalitions that can be formed while agents reach l_v can maximise $score(\boldsymbol{x}_v)$, as demonstrated by the following lemma.

Lemma 6.4 For the input subproblem, Algorithm 6.1 is guaranteed to converge to an optimal singleton solution.

Proof. Algorithm 6.1 finds a coalition of minimum size C^* that reaches l as fast as possible. This maximises the working time of the agents in C^*, that is, the number of positive decision variables, and therefore also the score (Equation 6.10). □

0	$(\underline{v_1, v_2}, v_3, v_4)$	**1**	8	$(\underline{v_1, v_4}, v_2, v_3)$	**9**	16	$(\underline{v_3, v_4}, v_1, v_2)$	**11**
1	$(\underline{v_2, v_1}, v_3, v_4)$	**2**	9	$(\underline{v_4, v_1}, v_2, v_3)$	**10**	17	$(\underline{v_4, v_3}, v_1, v_2)$	**12**
2	$(\underline{v_3, v_1}, v_2, v_4)$	**3**	10	(v_2, v_1, v_4, v_3)	10	~~18~~	~~(v_4, v_3, v_2, v_1)~~	~~12~~
3	$(\underline{v_1, v_3}, v_2, v_4)$	**4**	11	(v_1, v_2, v_4, v_3)	10	~~19~~	~~(v_3, v_4, v_2, v_1)~~	~~12~~
4	$(\underline{v_2, v_3}, v_1, v_4)$	**5**	12	(v_1, v_3, v_4, v_2)	10	~~20~~	~~(v_2, v_4, v_3, v_1)~~	~~12~~
5	$(\underline{v_3, v_2}, v_1, v_4)$	**6**	13	(v_3, v_1, v_4, v_2)	10	~~21~~	~~(v_4, v_2, v_3, v_1)~~	~~12~~
6	$(\underline{v_4, v_2}, v_1, v_3)$	**7**	14	(v_4, v_1, v_3, v_2)	10	~~22~~	~~(v_3, v_2, v_4, v_1)~~	~~12~~
7	$(\underline{v_2, v_4}, v_1, v_3)$	**8**	15	(v_1, v_4, v_3, v_2)	10	~~23~~	~~(v_2, v_3, v_4, v_1)~~	~~12~~

Figure 6.1 Illustrative example of the execution of Algorithm 6.3 on a problem where $V = \{v_1, v_2, v_3, v_4\}$, $Prec = \emptyset$, and $\boldsymbol{x}_{v_1}$ is the only feasible solution, which is therefore optimal. Each cell of each column is an iteration of the loop at line 4, except the first cell of the first column, which represents the initial incumbent solution obtained at line 3. Each cell reports the iteration number, current schedule, and number of schedules starting with distinct 2-permutations of V investigated so far. The schedules are generated using Heap's method [Sedgewick 1977]. Let $\boldsymbol{x}_{v_1}$ be found at iteration 0. ANT takes 17 iterations before ensuring that, for each 2-permutation of V p, at least one schedule starting with p has been investigated. Thus, the last 6 schedules (25% of the total) are skipped.

Corollary 6.5 For the input schedule, Algorithm 6.2 is guaranteed to converge to an optimal solution.

Hence, if we were to perform the tasks according to a given order V^α, Algorithm 6.2 would find a solution $\boldsymbol{x}^\alpha$ with maximum score, in particular due to lines $4-6$ (Section 6.2.1). The previous results lead to the following theorem.

Theorem 6.6 Algorithm 6.3 is exact.

Proof. Corollary 6.5 implies that Algorithm 6.3 finds an optimal solution to each possible schedule (line 5). Hence, we only need to show that the stopping criterion (line 7) does not exclude schedules that have to be investigated. This follows from the discussion in Section 6.2.1: if we investigated all schedules involving k tasks, without finding a solution, then the maximum possible number of completed tasks must be less than k, thus there is no need to continue. □

Lastly, Algorithm 6.3 is *pleasingly* parallel [Kepner 2009, Section 2.1] because each possible schedule (line 6) is investigated in an independent thread, and anytime since each iteration of the loop at line 4 updates the incumbent solution.

6.2.3 Computational Complexity

Algorithm 6.1 requires $O(|A| \log |A|)$ time for the sorting at line 2 [Cormen et al. 2009], and $\bar{a} = O(|A| \cdot (\gamma_v - \alpha_v))$ time and space to define when each agent $a \in C^*$ is working on v. Its

total time complexity is $\bar{b} = \mathrm{O}(|\mathrm{A}| \log |\mathrm{A}| \cdot t_{max})$. Algorithm 6.2 requires $\bar{c} = \mathrm{O}(|\mathrm{V}| \cdot |\mathrm{L}| \cdot \bar{b})$ time and $\bar{d} = \mathrm{O}(|\mathrm{A}| \cdot t_{max})$ space to determine when and where each agent $a \in \mathrm{A}$ works, since each variable $y_{v,l}$ can be stored in constant space (e.g., using a fixed-size string).

Algorithm 6.3 requires: $\mathrm{O}(|\mathrm{V}| \log |\mathrm{V}|)$ time and $\mathrm{O}(|\mathrm{V}|)$ space for line 2; $\bar{c}$ time and $\bar{d}$ space for line 3; $\mathrm{O}(|\mathrm{V}|!)$ time for the loop at line 4, and $\mathrm{O}(|\mathrm{V}|)$ time for checking the condition at line 5. The condition at line 7 can be checked in constant time and space using counter variables. Hence, ANT has an overall time complexity of:

$$\mathrm{O}\left(|\mathrm{V}| \log |\mathrm{V}| + |\mathrm{V}|! \cdot (|\mathrm{V}| + \bar{c})\right) = \mathrm{O}\left(|\mathrm{V}|! \cdot |\mathrm{L}| \cdot |\mathrm{A}| \log |\mathrm{A}| \cdot t_{max}\right) \tag{6.22}$$

and an overall space complexity of:

$$\mathrm{O}\left(|\mathrm{V}| + (1 + \kappa) \cdot \bar{d}\right) = \mathrm{O}\left(|\mathrm{V}| + \kappa \cdot |\mathrm{A}| \cdot t_{max}\right) \tag{6.23}$$

where κ is the maximum number of threads available. From Lemma 6.3 and Theorem 6.6, it follows that no exact MARSC algorithm can have a time and space complexity lower than that specified by Equations 6.22 and 6.23.

6.3 Empirical Evaluation

We wrote a test framework in Java[2] consisting of two suites. The first is an extension of Section 4.3, while the second is a variant of Section 5.4 with an extended dataset[3] of 567492 task demands (+67%) generated from LFB records of the last 12 years. We used the first suite to test ANT on small-scale synthetic problems, and the second suite to test an approximate variant called ANT-ε on large-scale realistic problems. This variant simply limits to ε the maximum number of iterations done at line 4 in Algorithm 6.3. We used Heap's method [Sedgewick 1977] to generate task permutations[4] in Algorithm 6.3, and set $\varepsilon = 10^5$. As baselines, we used CTS (Section 4.2), augmented to solve the MARSC, and EDF[5], equivalent to lines 2 and 3 of Algorithm 6.3. Below, we detail our test suites and discuss the results.

[2] https://doi.org/10.5281/zenodo.5375844

[3] https://doi.org/10.5281/zenodo.4018139

[4] In preliminary tests, it proved to be the fastest in finding anytime solutions among the methods reported in [Sedgewick 1977].

[5] We use it in place of TOPTW algorithms, which ignore coalition formation and real-time domains.

6.3.1 Setup

Let $\mathcal{N}$ and $\mathcal{U}$ denote the normal and uniform distribution, respectively. A test configuration consists of the following base parameters:

- $|\mathrm{V}| = |\mathrm{A}| \cdot k$, where $k \in \{1, \dots, 20\}$;
- For each task v_i, with $i \leq |\mathrm{V}| - 1$, if $\alpha_{v_i} \leq \alpha_{v_{i+1}}$ and $\gamma_{v_i} < \gamma_{v_{i+1}}$, then $v_i \prec v_{i+1}$ has probability 0.5. That is, the task order is a partial chain defined by coin tosses.

Each suite then uses the following parameters.

Suite 1: **Synthetic Problems**

- $|\mathrm{A}| = 2$. Consequently, problems have up to 40 tasks. From Equation 6.22, it follows that a problem contains up to $40! \approx 8.16 \cdot 10^{47}$ schedules to be investigated;
- The location space is a 50×50 grid. $\forall a \in \mathrm{A}$, $\forall l_1, l_2 \in \mathrm{L}$, $\rho(a, l_1, l_2)$ is the Manhattan distance between l_1 and l_2 divided by the speed of a, which is sampled from $\mathcal{U}(1, 2)$. Each agent has a random initial location, while each task has 2 random possible locations;
- $\forall v \in \mathrm{V}$, $w_v \sim \mathcal{U}(10, 50)$, $\alpha_v \sim \mathcal{U}(5, 600)$, $\gamma_v \sim \mathcal{U}(\alpha_v, 600)$, $\beta_v \sim \mathcal{U}(\alpha_v, \gamma_v)$, and $\phi_v \sim \mathcal{U}(1, 2)$.

Suite 2: **LFB Problems**

- Each task demand is defined by a record dated between 1 January 2009 and 31 May 2021 as follows: $\alpha_v = \min_{a \in \mathrm{A}} \rho(a, l_a^0, l_v)$; $\gamma_v = \alpha_v + \kappa$, where κ is the attendance time (in seconds) of the firefighters; $\beta_v \sim \mathcal{U}(\alpha_v, \gamma_v)$; $|\mathrm{L}_v| = 1$, and $\phi_v = 1$. Lastly, since the median attendance time in the whole dataset is about 5 minutes, we set $w_v \sim \mathcal{U}(10, 300)$ to simulate wide-ranging workloads;
- The remaining parameters are the same as in Section 5.4.1.

Coalition Value Distributions We use UC_NDCS and UC_Agent-based from Section 5.4.1, with the following addition. To simulate real-time domains where the further away l_v is, the lower the benefit of performing v [Stankovic et al. 2013], we decrease each $\mu_v = u(\mathrm{C}, v, l)$ by $z \sim \mathcal{U}(\mu_v/10, \mu_v/4)$ with probability $\rho(\hat{a}, l_{\hat{a}}, l_v)/(t_{max} + 1)$, where $\hat{a}$ is the last agent to reach l_v, and $l_{\hat{a}}$ is the location of the task previously completed by $\hat{a}$, or $l_{\hat{a}}^0$ otherwise. Since it is a touchstone for the study of coalition formation problems [Sandholm, Larson et al. 1999], we also use the following special case of Superadditive distribution: $u(\mathrm{C}, v, l) = |\mathrm{C}|$. We ensured consistency between the results of the algorithms as follows. Regarding Superadditive, all coalition values

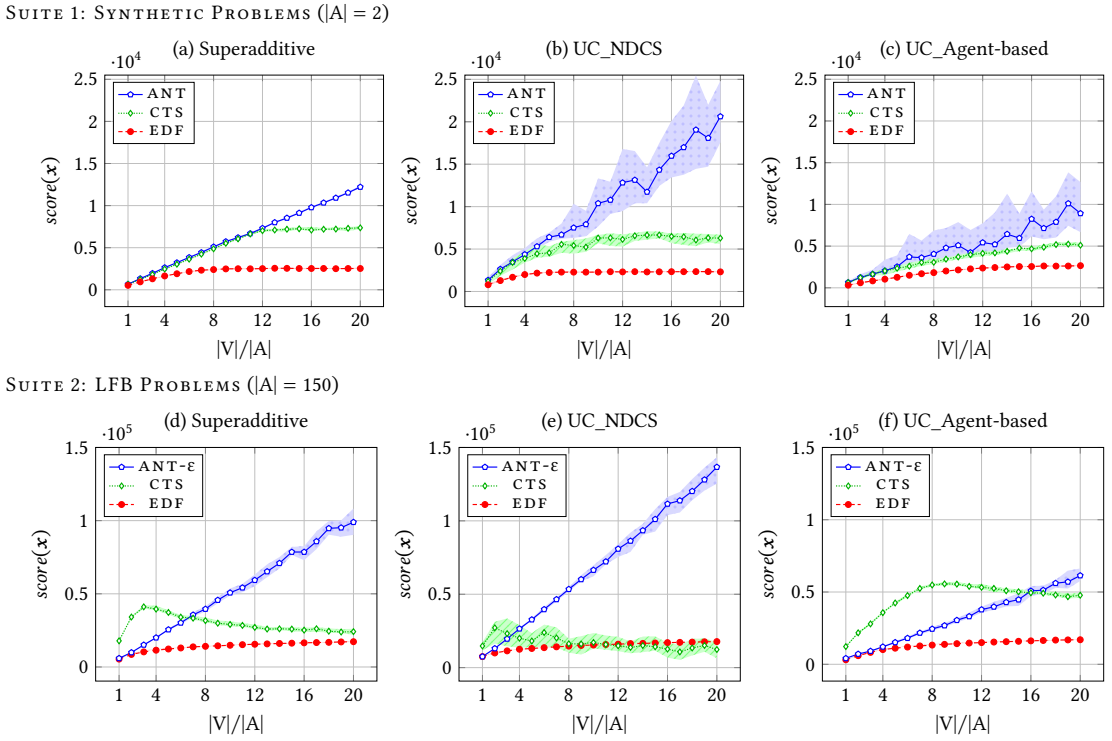


Figure 6.2 Performance of ANT and ANT-ε in our test suites. Each subfigure contains tests performed with the coalition value distribution in the title. Each point is the median and 95% confidence interval over 100 replicates. The X-axis indicates task-to-agent ratios, while the Y-axis reports solution scores (Equation 6.10). The higher the scores, the better the solutions.

were computed and stored in hash maps before running the tests. As it is only defined by coalition sizes, this preprocessing step took O(|A|) time. For UC_NDCS and UC_Agent-based, the hash maps were lazy-initialised and shared among all problems (as done in Section 5.4.1).

For each test suite, algorithm and coalition value, we solved 100 problems, and measured the median and 95% confidence interval of solution score (Equation 6.10) and CPU time[6]. Since we have 3 algorithms, 20 task-to-agent ratios, 3 coalition value distributions, 100 replicates, and 2 test suites, the total number of tests performed is 36000.

6.3.2 Results

Figure 6.2 reports the results of our tests. The solution score generally increases because it is an absolute metric, hence the larger the problem size, the more tasks there are that can be completed by their soft latest times. Let $\eta^{\Lambda}_{\mathrm{B}} = score(\boldsymbol{x}_{\Lambda})/score(\boldsymbol{x}_{\mathrm{B}})$ be the *performance improvement* of $\Lambda \in \{\text{ANT}, \text{ANT-}\varepsilon\}$ over $\mathrm{B} \in \{\text{CTS}, \text{EDF}\}$. Regarding Suite 1 (synthetic problems, $|\mathrm{A}| = 2$), being an exact algorithm, ANT dominated both baselines. More precisely, we recorded the following median improvements:

- Superadditive (Figure 6.2a): $\eta^{\text{ANT}}_{\text{CTS}} \approx 1.07 \pm [0.57, 0.06]$ and $\eta^{\text{ANT}}_{\text{EDF}} \approx 1.36 \pm [0.75, 0.32]$;

[6]Based on an Intel Xeon Gold 6138 (2 GHz, 40 threads).

- UC_NDCS (Figure 6.2b): $\eta_{CTS}^{ANT} \approx 1.94 \pm [1.79, 0.81]$ and $\eta_{EDF}^{ANT} \approx 1.92 \pm [1.39, 0.82]$;
- UC_Agent-based (Figure 6.2c): $\eta_{CTS}^{ANT} \approx 1.89 \pm [0.64, 0.82]$ and $\eta_{EDF}^{ANT} \approx 2.35 \pm [1.02, 0.95]$.

The overall median is $1.9 \pm [1.83, 0.89]$. The wider confidence intervals in Figures 6.2b and 6.2c are due to the perturbations introduced by the UC factor. Regarding Suite 2 (LFB problems, $|A| = 150$), the median improvements are:

- Superadditive (Figure 6.2d): $\eta_{CTS}^{ANT\text{-}\varepsilon} \approx 1.83 \pm [2.27, 1.55]$ and $\eta_{EDF}^{ANT\text{-}\varepsilon} \approx 3.38 \pm [2.04, 2.37]$;
- UC_NDCS (Figure 6.2e): $\eta_{CTS}^{ANT\text{-}\varepsilon} \approx 4.23 \pm [6.86, 3.79]$ and $\eta_{EDF}^{ANT\text{-}\varepsilon} \approx 4.26 \pm [3.05, 3.25]$;
- UC_Agent-based (Figure 6.2f): $\eta_{CTS}^{ANT\text{-}\varepsilon} \approx 0.58 \pm [0.70, 0.33]$ and $\eta_{EDF}^{ANT\text{-}\varepsilon} \approx 2.15 \pm [1.42, 1.13]$.

The overall median is $2.6 \pm [8.49, 2.5]$. The confidence intervals are less wide in Suite 2 because the tasks are chosen in chronological order, thus the problems vary less in difficulty. The non-monotonic performance of CTS depends both on its strategy of prioritising urgent and uncompleted tasks (Section 4.2.2), and on the fact that the tasks are sorted chronologically. After reaching the 'peak' performance (e.g., $|V|/|A| = 9$ in Figure 6.2f), CTS becomes increasingly ineffective because, when there are too many chronologically ordered tasks, performing urgent and uncompleted tasks first is similar to EDF. Furthermore, the reason why it has the worst performance for $|V|/|A| \geq 12$ in Figure 6.2e is that it minimises the number of coalition allocations, which is penalised by any distribution based on NDCS. On the other hand, ANT-ε tends to be more effective as the problem size increases, and is also more consistent, both with distributions such as Superadditive and UC_Agent-based, which reward larger coalitions more and thus are suitable for real-time domains like disaster response, and with (a variant of) NDCS, which is more difficult to prune [Rahwan, Ramchurn et al. 2009, Section 5.2].

In Suite 1, CTS and EDF ran in less than 1 ms, while ANT had a median of $15.56 \pm [8.9, 15.55]$ minutes. In Suite 2, the medians were: $78 \pm [64, 72]$ ms for EDF; $18.35 \pm [39.7, 18.08]$ seconds for CTS, and $5.03 \pm [3.84, 4.44]$ seconds for ANT-ε. Hence, ANT-ε was typically 3.65 times faster than CTS. The total RAM usage was 7.25 GB for Suite 1, and 20.4 GB for Suite 2.

To summarise, ANT solved synthetic problems with 2 agents and up to 40 tasks in less than 25 minutes, finding up to 3.73 times better solutions than our incomplete baselines. On the other hand, compared to the state-of-the-art CFSTP algorithm, ANT-ε found 2.6 times better median solutions in less than a third of the time.

Conclusions

In Chapter 4, we proposed two novel algorithms to solve the CFSTP. The first is CFLA2, an improved version of CFLA, and the second is CTS, which is the first to be simultaneously anytime, efficient and with convergence guarantee. CFLA2 can replace CFLA in solving static or small-scale problems, while CTS provides a baseline for benchmarks with dynamic and large-scale problems. Moreover, we showed how a simplified but parallel and faster variant of CTS is enough to compete with high-performance algorithms such as BinaryMS and DSA in the RCRS. Because it significantly outperforms CFLA, and is more applicable than CFLA2, we can consider CTS to be the new state-of-the-art CFSTP algorithm. Thanks to its features (Section 4.2.3), CTS can also be used in contexts other than disaster response, but which can be still characterised by the CFSTP model, such as multi-robot area coverage, or exploration of environments that are dangerous for humans [Ramchurn, Polukarov et al. 2010, Section 8].

Chapter 5 has focused on eliminating the major gaps in the CFSTP literature. Specifically, we gave a novel mathematical programming formulation, which is significantly shorter than the original (Section 3.3). Then, by reducing the CFSTP to a DynDCOP, we designed D-CTS, the first distributed version of CTS. Lastly, using real-world and open data provided by the LFB, and a large-scale test framework, we compared D-CTS with DSA-SDP, a state-of-the-art distributed algorithm. In situations where the number of agents monotonically decreases over time, D-CTS has slightly better median performance, as well as significantly lower communication overhead and time complexity. D-CTS sets the first large-scale, dynamic and distributed CFSTP benchmark.

Finally, in Chapter 6, to meet all our research objectives (Section 1.3), we defined the MARSC, a generalisation of both the CFSTP and the TOPTW that can be used in real-time domains. Moreover, we proposed ANT, the first anytime, exact and parallel MARSC algorithm, as well as an approximate variant called ANT-ε. ANT is also the first exact algorithm for the CFSTP. Using extended versions of the test frameworks of Chapters 4 and 5, we showed that ANT can solve problems with 2 agents and up to 40 tasks in less than 25 minutes, while ANT-ε can solve problems with 150 agents and up to 3000 tasks in at most 8.87 seconds. ANT can be used in applications where optimal solutions are required and time is not a limiting factor, such as industrial scheduling and timetabling problems [Petcu and Faltings 2005a], or routing protocols for static sensor networks [Kho, Rogers and Jennings 2009]. On the other hand, ANT-ε can be

used in large-scale and dynamic environments where it is important to minimise communication overhead [Chapman, Rogers, Jennings and Leslie 2011; Ponda et al. 2015].

Main Limitations of Our Work

We cannot define the quality of the approximation obtained by (D-)CTS in general settings (Section 4.2.3). Moreover, the fact that it maximises the working time of agents (Section 4.3) implies that some agents may take longer to complete some tasks and therefore may not work on others. Thus, if an optimal solution exists, (D-)CTS is not guaranteed to find it.

Although it generalises the CFSTP and the TOPTW with important constraints and a more expressive objective function, the MARSC assumes that the agents have total knowledge, and that the environment has a deterministic behaviour (Section 2.3). In real-world domains, the status of the tasks may be partially known or unknown, while exogenous events, such as unexpected failures or severe weather conditions, may reduce the ability of agents to perform them. Thus, agents have to balance the *exploration* of the environment with the *exploitation* of the information acquired [Taylor, Jain, Y. Jin et al. 2010; Taylor, Jain, Tandon et al. 2011].

The performance of ANT depends primarily on the order in which the schedules are investigated (line 4 in Algorithm 6.3). For example, if in Figure 6.1 we had investigated (v_3, v_4, v_1, v_2) and (v_4, v_3, v_1, v_2) at iterations 10 and 11, respectively, then we would have skipped the last 12 schedules (50% of the total), thus minimising the number of iterations required. Moreover, although ANT-ε is an approximate variant, it is not clear what the relationship is between the value of ε and the quality of the solutions obtained.

Possible Future Work

In its current state, the field of multi-agent optimisation is very much focused on academic testbeds, and little on real-world scenarios [V. Dignum and F. Dignum 2020; Zivan 2021]. In particular, research on DCOPs is mostly concerned with creating effective algorithms for solving well-known benchmark problems[7], which do not necessarily reflect realistic situations. Consequently, possible future work should not only seek to mitigate the above limitations, but also do so with the aim of overcoming the current stagnation in the DCOP research community.

A first objective is to extend our LFB test framework (Section 5.4) by:

1. Adding other state-of-the-art distributed algorithms as baselines, such as DALO [Kiekintveld et al. 2010], Bounded FMS [Macarthur, Farinelli et al. 2010], SBDO [Billiau, Chang and Ghose 2012], GDBA [Okamoto, Zivan, Nahon et al. 2016], D-Gibbs [D. T. Nguyen et al. 2019], and FMC_TA [Nelke, Okamoto and Zivan 2020; Tkach and Amador 2021];

[7] For example, those generated with the FRODO framework [Fioretto, Pontelli and Yeoh 2018, Section 9.4.2].

2. Developing more realistic coalition value distributions, perhaps by extending the ones identified in [Changder, Aknine and Animesh Dutta 2021, Section 5.1];

3. Also studying *exploration* scenarios [Fioretto, Pontelli and Yeoh 2018], that is, designing tests in which tasks are gradually added to the system;

4. Defining a travel time function suitable for ground vehicles. More precisely, the current definition of $\rho(\cdot)$ is based on the shortest distance between two points on the surface of a sphere (Section 5.4.1). This is only suitable for multi-UAV systems flying high enough not to have to avoid collisions. However, already with such a simple function, D-CTS is one order of magnitude more efficient than DSA-SDP (Section 5.4.2). Consequently, it would be worth studying how the results change with a function that considers more realistic factors, such as terrain type or road congestion.

The concept of k-optimality [Farinelli, Vinyals et al. 2013; Service and Adams 2011] could be used to design extensions of D-CTS and ANT-ε with provable bounds on solution quality. Moreover, given its advantages (Section 5.3) and the scarcity of incomplete DynDCOP algorithms (Section 2.3.2), another important objective is to create a D-CTS variant able to solve general DynDCOPs.

The nascent paradigm of enhancing multi-agent optimisation with machine learning has already proved capable of improving the performance of VRP algorithms considerably [Bai et al. 2021]. Motivated by this, we could use Q-learning to create a more effective method for investigating schedules in ANT, and combine it with a CUDA implementation to establish trade-offs between computational power and problem size. This improved version of ANT could be validated against general meta-heuristics that are also based on permutations of problem elements, such as Squeaky Wheel Optimisation [Joslin and Clements 1999]. Since the MARSC generalises the TOPTW (Theorem 6.2), we could evaluate a simplified version of ANT in well-established TOPTW benchmarks. To the best of our knowledge, despite being the most studied vehicle routing problem [Vansteenwegen and Gunawan 2019], so far only 4 exact TOPTW algorithms have been proposed [Boussier, Feillet and Gendreau 2007; Gedik et al. 2017; El-Hajj et al. 2015; Tae and B.-I. Kim 2015]. Hence, it would be interesting to compare ANT with these algorithms. Moreover, given that the MARSC is related to the CSG problem, which remains central to numerous multi-agent applications, following contributions like [Agarwal et al. 2015; Svensson et al. 2013; Ueda et al. 2010], ANT could also be adapted as a new anytime, parallel and exact CSG algorithm.

Finally, to capture problems where agents have limited perception, communication is imperfect or noisy, and the environment behaviour is stochastic (e.g., the location and severity of the fires are unknown, thus agents must scout the disaster area to find them), we could extend the reduction of the MARSC to a DynDCOP given in Section 6.1.5 with elements of autonomic communications theory [Dobson et al. 2006] and Probabilistic DCOPs [Fioretto, Pontelli and Yeoh 2018, Section 6]. To the best of our knowledge, this would result in the first DCOP model to cope with both dynamic and stochastic environments [Fioretto, Pontelli and Yeoh 2018, Section 9.1].

References

Agarwal, Manoj et al. (2015). 'Parallel multi-objective multi-robot coalition formation'. In: *Expert Systems with Applications* 42.21, pp. 7797–7811 (cit. on p. 89).

Alexander, David E. (2002). *Principles of Emergency Planning and Management.* Oxford University Press (cit. on pp. 1, 2).

Allen-Williams, Mair and Nicholas R. Jennings (2010). 'Bayesian Agent Adaptation in Complex Dynamic Systems'. In: *Handbook of Research on Complex Dynamic Process Management: Techniques for Adaptability in Turbulent Environments.* IGI Global, pp. 172–208 (cit. on pp. 17, 18).

Amato, Christopher, Girish Chowdhary et al. (2013). 'Decentralized Control of Partially Observable Markov Decision Processes'. In: *IEEE Conference on Decision and Control*, pp. 2398–2405 (cit. on p. 28).

Amato, Christopher, George Konidaris et al. (2016). 'Policy search for multi-robot coordination under uncertainty'. In: *International Journal of Robotics Research* 35.14, pp. 1760–1778 (cit. on p. 16).

Amato, Christopher and Frans Oliehoek (2015). 'Scalable Planning and Learning for Multiagent POMDPs'. In: *AAAI.* Vol. 29. 1 (cit. on pp. 17, 18).

Amorim, Junier Caminha, Vander Alves and Edison Pignaton de Freitas (2020). 'Assessing a swarm-GAP based solution for the task allocation problem in dynamic scenarios'. In: *Expert Systems with Applications* 152.113437 (cit. on p. 31).

Andrews, George E. and Kimmo Eriksson (2004). *Integer Partitions.* Cambridge University Press (cit. on p. 12).

Arif, Muhammad Usman (2021). 'Robot Coalition Formation Against Time-Extended Multi-Robot Tasks'. In: *International Journal of Intelligent Unmanned Systems (in press).* DOI: 10.1108/IJIUS-12-2020-0070 (cit. on p. 30).

Arshad, Muhammad and Marius C. Silaghi (2004). 'Distributed Simulated Annealing'. In: *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems* 112. Ed. by Zhang, Weixiong and Sorge, Volker, pp. 35–47 (cit. on p. 26).

Ayari, Emna, Sameh Hadouaj and Khaled Ghedira (2017). 'A Dynamic Decentralised Coalition Formation Approach for Task Allocation under Tasks Priority Constraints'. In: *International Conference on Advanced Robotics (ICAR).* IEEE, pp. 250–255 (cit. on p. 29).

Aziz, Haris et al. (2021). 'Multi-Robot Task Allocation–Complexity and Approximation'. In: arXiv: 2103.12370 [cs.RO] (cit. on p. 11).

Bai, Ruibin et al. (2021). 'Analytics and Machine Learning in Vehicle Routing Research'. In: arXiv: 2102.10012 [cs.LG] (cit. on p. 89).

Baker, Chris A. B. et al. (2016). 'Planning Search and Rescue Missions for UAV Teams'. In: *ECAI*, pp. 1777–1782 (cit. on pp. 4, 17, 18, 33).

Barambones, Jose, Ricardo Imbert and Cristian Moral (2021). 'Applicability of Multi-Agent Systems and Constrained Reasoning for Sensor-Based Distributed Scenarios: A Systematic Mapping Study on Dynamic DCOPs'. In: *Sensors* 21.3807 (cit. on pp. 26, 29).

Barbulescu, Laura et al. (2010). 'Distributed Coordination of Mobile Agent Teams: The Advantage of Planning Ahead'. In: *AAMAS*. Vol. 1, pp. 1331–1338 (cit. on p. 30).

Bazinin, Sagi and Guy Shani (2018). 'Iterative Planning for Deterministic QDec-POMDPs'. In: *EPiC Series in Computing* 55, pp. 15–28 (cit. on pp. 17, 18).

Becker, Raphen et al. (2004). 'Solving Transition Independent Decentralized Markov Decision Processes'. In: *JAIR* 22, pp. 423–455 (cit. on p. 16).

Bellman, Richard (2003). *Dynamic Programming*. Dover Publications Mineola (cit. on pp. 6, 15).

Bernstein, Daniel S. et al. (2002). 'The Complexity of Decentralized Control of Markov Decision Processes'. In: *Mathematics of Operations Research* 27.4, pp. 819–840 (cit. on pp. 15, 28).

Bertsekas, Dimitri P. (2012). *Dynamic Programming and Optimal Control*. Fourth edition. Athena Scientific (cit. on p. 15).

Billiau, Graham, Chee Fon Chang and Aditya Ghose (2012). 'SBDO: A New Robust Approach to Dynamic Distributed Constraint Optimisation'. In: *PRIMA*. Springer Berlin Heidelberg, pp. 11–26 (cit. on pp. 27, 88).

– (2014). 'Multi-objective Distributed Constraint Optimization Using Semi-rings'. In: *International Conference on Principles and Practice of Multi-Agent Systems*. Springer, pp. 407–422 (cit. on p. 27).

Bischoff, Esther et al. (2020). 'Multi-Robot Task Allocation and Scheduling Considering Cooperative Tasks and Precedence Constraints'. In: *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 3949–3956 (cit. on p. 30).

Bistaffa, Filippo, Alessandro Farinelli, Jesús Cerquides et al. (2017). 'Algorithms for Graph-Constrained Coalition Formation in the Real World'. In: *ACM Transactions on Intelligent Systems and Technology* 8.4, p. 60 (cit. on p. 13).

Bistaffa, Filippo, Alessandro Farinelli, Georgios Chalkiadakis et al. (2017). 'A cooperative game-theoretic approach to the social ridesharing problem'. In: *AIJ* 246, pp. 86–117 (cit. on p. 13).

Bistaffa, Filippo, Alessandro Farinelli and Sarvapali D. Ramchurn (2015). 'Sharing Rides with Friends: A Coalition Formation Algorithm for Ridesharing'. In: *AAAI* (cit. on p. 12).

Bogner, Karin et al. (2018). 'Optimised scheduling in human–robot collaboration–a use case in the assembly of printed circuit boards'. In: *International Journal of Production Research* 56.16, pp. 5522–5540 (cit. on p. 31).

Bogomolnaia, Anna and Matthew O. Jackson (2002). 'The Stability of Hedonic Coalition Structures'. In: *Games and Economic Behavior* 38.2, pp. 201–230 (cit. on p. 14).

Boussier, Sylvain, Dominique Feillet and Michel Gendreau (2007). 'An exact algorithm for team orienteering problems'. In: *4OR* 5.3, pp. 211–230 (cit. on p. 89).

Brucker, Peter (2007). *Scheduling Algorithms*. Fifth edition. Springer-Verlag (cit. on p. 4).

Busoniu, Lucian et al. (2017). *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC press (cit. on p. 15).

Calvaresi, Davide et al. (2021). 'Real-time multi-agent systems: rationality, formal model, and empirical results'. In: *JAAMAS* 35.1, pp. 1–37 (cit. on p. 4).

Cao, Yongcan et al. (2013). 'An Overview of Recent Progress in the Study of Distributed Multi-agent Coordination'. In: *IEEE Transactions on Industrial informatics* 9.1, pp. 427–438 (cit. on p. 11).

Capitan, Jesus et al. (2013). 'Decentralized multi-robot cooperation with auctioned POMDPs'. In: *International Journal of Robotics Research* 32.6, pp. 650–671 (cit. on pp. 16, 28).

Cerquides, Jesus et al. (2013). 'A Tutorial on Optimization for Multi-Agent Systems'. In: *The Computer Journal* 57.6, pp. 799–824 (cit. on pp. 6, 11, 29).

Chalkiadakis, Georgios and Craig Boutilier (2004). 'Bayesian Reinforcement Learning for Coalition Formation under Uncertainty'. In: *AAMAS*. Vol. 3, pp. 1090–1097 (cit. on p. 13).

– (2012). 'Sequentially optimal repeated coalition formation under uncertainty'. In: *JAAMAS* 24.3, pp. 441–484 (cit. on p. 13).

Chalkiadakis, Georgios, Edith Elkind, Evangelos Markakis et al. (2010). 'Cooperative Games with Overlapping Coalitions'. In: *JAIR* 39, pp. 179–216 (cit. on p. 13).

Chalkiadakis, Georgios, Edith Elkind and Michael Wooldridge (2012). *Computational Aspects of Cooperative Game Theory*. Morgan & Claypool Publishers (cit. on pp. 34, 55).

Changder, Narayan, Samir Aknine and Animesh Dutta (2021). 'Improving coalition structure search with an imperfect algorithm: analysis and evaluation results'. In: *Artificial Intelligence Review* 54.1, pp. 397–425 (cit. on p. 89).

Chapman, Archie C., Rosa Anna Micillo et al. (2010). 'Decentralized Dynamic Task Allocation Using Overlapping Potential Games'. In: *The Computer Journal* 53.9, pp. 1462–1477 (cit. on p. 6).

Chapman, Archie C., Alex Rogers and Nicholas R. Jennings (2011). 'Benchmarking hybrid algorithms for distributed constraint optimisation games'. In: *JAAMAS* 22.3, pp. 385–414 (cit. on p. 6).

Chapman, Archie C., Alex Rogers, Nicholas R. Jennings and David S. Leslie (2011). 'A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems'. In: *The Knowledge Engineering Review* 26.4, pp. 411–444 (cit. on pp. 6, 88).

Chen, Yuxiao, Ugo Rosolia and Aaron D. Ames (2021). 'Decentralized Task and Path Planning for Multi-Robot Systems'. In: *IEEE Robotics and Automation Letters* 6.3, pp. 4337–4344 (cit. on p. 17).

Chen, Ziyu, Yanchen Deng and Tengfei Wu (2017). 'An Iterative Refined Max-sum_AD Algorithm via Single-side Value Propagation and Local Search'. In: *AAMAS*, pp. 195–202 (cit. on p. 24).

Chen, Ziyu, Xingqiong Jiang et al. (2019). 'A Generic Approach for Accelerating Belief Propagation based DCOP Algorithms via A Branch-and-Bound Technique'. In: *AAAI*. Vol. 33. 1, pp. 6038–6045 (cit. on p. 27).

Chen, Ziyu, Lizhen Liu et al. (2020). 'A genetic algorithm based framework for local search algorithms for distributed constraint optimization problems'. In: *JAAMAS* 34, pp. 1–31 (cit. on pp. 21, 25).

Chen, Ziyu, Tengfei Wu et al. (2018). 'An Ant-Based Algorithm to Solve Distributed Constraint Optimization Problems'. In: *AAAI* 32.1 (cit. on pp. 21, 25).

Choi, Charles Q. (2021). *7 Revealing Ways AIs Fail*. `https://spectrum.ieee.org/ai-failures`. (Visited on 01/11/2021) (cit. on p. 14).

Cohen, Liel and Roie Zivan (2017). 'Max-sum Revisited: The Real Power of Damping'. In: *AAMAS*, pp. 111–124 (cit. on p. 24).

Coppola, Damon P. (2006). *Introduction to International Disaster Management*. Elsevier (cit. on p. 2).

Cormen, Thomas H. et al. (2009). *Introduction to Algorithms*. Third edition. MIT press (cit. on pp. 20, 22, 49, 67, 82).

Czarnecki, Emily and Ayan Dutta (2021). 'Scalable hedonic coalition formation for task allocation with heterogeneous robots'. In: *Intelligent Service Robotics* 3, pp. 501–517 (cit. on p. 14).

Dadvar, Mehdi and Soheil Habibian (2021). 'Contemporary Research Trends in Response Robotics'. In: arXiv: `2105.07812 [cs.CY]` (cit. on pp. 3, 29).

Dechter, Rina (2003). *Constraint Processing*. Morgan Kaufmann (cit. on p. 6).

Dehghani, Mostafa et al. (2021). 'The Benchmark Lottery'. In: arXiv: `2107.07002 [cs.LG]` (cit. on p. 14).

Delle Fave, Francesco Maria, Alessandro Farinelli et al. (2012). 'A Methodology for Deploying the Max-Sum Algorithm and a Case Study on Unmanned Aerial Vehicles'. In: *IAAI* (cit. on p. 24).

Delle Fave, Francesco Maria, Alex Rogers et al. (2012). 'Deploying the Max-Sum Algorithm for Decentralised Coordination and Task Allocation of Unmanned Aerial Vehicles for Live Aerial Imagery Collection'. In: *International Conference on Robotics and Automation*. IEEE, pp. 469–476 (cit. on p. 24).

Delle Fave, Francesco Maria, Zhe Xu et al. (2010). 'Decentralised Coordination of Unmanned Aerial Vehicles for Target Search using the Max-Sum Algorithm'. In: *AAMAS Workshop on Agents in Real Time and Dynamic Environment*, pp. 35–44 (cit. on p. 24).

Dias, M. Bernardine et al. (2006). 'Market-Based Multirobot Coordination: A Survey and Analysis'. In: *Proceedings of the IEEE* 94.7, pp. 1257–1270 (cit. on pp. 28, 68).

Diederich, Adele (2001). 'Sequential Decision Making'. In: *International Encyclopedia of the Social & Behavioral Sciences*. Ed. by Smelser, Neil J. and Baltes, Paul B. Oxford: Pergamon, pp. 13917–13922 (cit. on pp. 6, 14).

Dignum, Virginia and Frank Dignum (2020). 'Agents Are Dead. Long Live Agents!' In: *AAMAS*, pp. 1701–1705 (cit. on p. 88).

Dobson, Simon et al. (2006). 'A Survey of Autonomic Communications'. In: *ACM TAAS* 1.2, pp. 223–259 (cit. on p. 89).

Dorigo, Marco, Mauro Birattari and Thomas Stutzle (2006). 'Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique'. In: *IEEE Computational Intelligence Magazine* 1.4, pp. 28–39 (cit. on p. 25).

Dos Santos, Fernando and Ana L. C. Bazzan (2011). 'Towards efficient multiagent task allocation in the RoboCup Rescue: a biologically-inspired approach'. In: *JAAMAS* 22.3, pp. 465–486 (cit. on p. 31).

Epstein, Daniel and Ana L. C. Bazzan (2011). 'Dealing with Coalition Formation in the RoboCup Rescue - An Euristic Approach'. In: *ICAART*. Vol. 2. SciTePress, pp. 717–720 (cit. on p. 3).

Farinelli, Alessandro, Manuele Bicego et al. (2017). 'A Hierarchical Clustering Approach to Large-scale Near-optimal Coalition Formation with Quality Guarantees'. In: *Engineering Applications of Artificial Intelligence* 59, pp. 170–185 (cit. on p. 14).

Farinelli, Alessandro, Luca Iocchi and Daniele Nardi (2004). 'Multirobot Systems: A Classification Focused on Coordination'. In: *IEEE Transactions on Systems, Man, and Cybernetics* 34.5, pp. 2015–2028 (cit. on p. 11).

Farinelli, Alessandro, Alex Rogers et al. (2008). 'Decentralised Coordination of Low-Power Embedded Devices Using the Max-Sum Algorithm'. In: *AAMAS*. Vol. 2, pp. 639–646 (cit. on p. 24).

Farinelli, Alessandro, Meritxell Vinyals et al. (2013). 'Distributed Constraint Handling and Optimization'. In: *Multiagent Systems*. Ed. by Weiss, Gerhard. Second edition. MIT Press. Chap. 12 (cit. on pp. 18, 20, 24, 89).

Feo-Flushing, Eduardo, Luca Maria Gambardella and Gianni A. Di Caro (2021). 'Spatially-Distributed Missions With Heterogeneous Multi-Robot Teams'. In: *IEEE Access* 9, pp. 67327–67348 (cit. on p. 30).

Ferreira, Barbara Arbanas, Tamara Petrović and Stjepan Bogdan (2021). 'Distributed Mission Planning of Complex Tasks for Heterogeneous Multi-Robot Teams'. In: arXiv: `2109.10106 [cs.RO]` (cit. on p. 30).

Ferreira, Paulo R., Felipe S. Boffo and Ana L. C. Bazzan (2007). 'Using Swarm-GAP for Distributed Task Allocation in Complex Scenarios'. In: *AAMAS*. Springer, pp. 107–121 (cit. on p. 31).

Ferreira, Paulo Roberto et al. (2010). 'RoboCup Rescue as multiagent task allocation among teams: experiments with task interdependencies'. In: *JAAMAS* 20.3, pp. 421–443 (cit. on p. 31).

Fioretto, Ferdinando, Enrico Pontelli and William Yeoh (2018). 'Distributed Constraint Optimization Problems and Applications: A Survey'. In: *JAIR* 61, pp. 623–698 (cit. on pp. 4, 6–8, 18–21, 23, 24, 26, 29, 60, 64, 67, 68, 88, 89).

Flammini, Michele et al. (2018). 'Online Coalition Structure Generation in Graph Games'. In: *AAMAS*, pp. 1353–1361 (cit. on p. 13).

Gallud, Ximo and Daniel Selva (2018). 'Agent-based simulation framework and consensus algorithm for observing systems with adaptive modularity'. In: *Systems Engineering* 21.5, pp. 432–454 (cit. on p. 31).

Gedik, Ridvan et al. (2017). 'A Constraint Programming Approach for the Team Orienteering Problem with Time Windows'. In: *Computers & Industrial Engineering* 107, pp. 178–195 (cit. on p. 89).

Geman, Stuart and Donald Geman (1993). 'Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images'. In: *Journal of Applied Statistics* 20.5-6, pp. 25–62 (cit. on p. 23).

Geramifard, Alborz et al. (2011). 'Online Discovery of Feature Dependencies'. In: *ICML*, pp. 881–888 (cit. on p. 15).

Gerkey, Brian P. and Maja J. Matarić (2004). 'A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems'. In: *International Journal of Robotics Research* 23.9, pp. 939–954 (cit. on pp. 1, 28, 30, 46).

Gershman, Amir, Amnon Meisels and Roie Zivan (2009). 'Asynchronous Forward Bounding for Distributed COPs'. In: *JAIR* 34, pp. 61–88 (cit. on p. 21).

Gershman, Samuel J., Eric J. Horvitz and Joshua B. Tenenbaum (2015). 'Computational rationality: A converging paradigm for intelligence in brains, minds, and machines'. In: *Science* 349.6245, pp. 273–278 (cit. on p. 28).

Ghassemi, Payam and Souma Chowdhury (2021). 'Multi-Robot Task Allocation in Disaster Response: Addressing Dynamic Tasks with Deadlines and Robots with Range and Payload Constraints'. In: *Robotics and Autonomous Systems (in press)*. DOI: 10.1016/j.robot.2021.103905 (cit. on p. 30).

Ghavamzadeh, Mohammad et al. (2015). 'Bayesian Reinforcement Learning: A Survey'. In: *Foundations and Trends in Machine Learning* 8.5-6, pp. 359–492 (cit. on p. 13).

Gini, Maria (2017). 'Multi-Robot Allocation of Tasks with Temporal and Ordering Constraints'. In: *AAAI*, pp. 4863–4869 (cit. on p. 11).

Godoy, Julio and Maria Gini (2013). 'Task Allocation for Spatially and Temporally Distributed Tasks'. In: *IAS*. Springer Berlin Heidelberg, pp. 603–612 (cit. on pp. 30, 31).

Goldman, Claudia V. and Shlomo Zilberstein (2003). 'Optimizing Information Exchange in Cooperative Multi-agent Systems'. In: *AAMAS*, pp. 137–144 (cit. on p. 15).

– (2004). 'Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis'. In: *JAIR* 22, pp. 143–174 (cit. on p. 15).

Grinshpoun, Tal and Amnon Meisels (2008). 'Completeness and Performance of the APO Algorithm'. In: *JAIR* 33, pp. 223–258 (cit. on p. 21).

Griva, Igor, Stephen G. Nash and Ariela Sofer (2009). *Linear and Nonlinear Optimization*. Second edition. SIAM (cit. on pp. 39, 59).

Guerrero, Jose, Gabriel Oliver and Oscar Valero (2017). 'Multi-robot Coalitions Formation with Deadlines: Complexity Analysis and Solutions'. In: *PloS one* 12.1 (cit. on p. 69).

Guestrin, Carlos, Daphne Koller and Ronald Parr (2002). 'Multiagent Planning with Factored MDPs'. In: *Advances in Neural Information Processing Systems*, pp. 1523–1530 (cit. on p. 14).

El-Hajj, Racha et al. (2015). 'A column generation algorithm for the team orienteering problem with time windows'. In: *The 45th International Conference on Computers & Industrial Engineering (CIE45)* (cit. on p. 89).

Harvey, Peter, Chee Fon Chang and Aditya Ghose (2006). 'Support-Based Distributed Search: A New Approach for Multiagent Constraint Processing'. In: *International Workshop on Argumentation in Multi-Agent Systems*. Springer, pp. 91–106 (cit. on p. 27).

Hawe, Glenn I. et al. (2012). 'Agent-Based Simulation for Large-Scale Emergency Response: A Survey of Usage and Implementation'. In: *ACM CSUR* 45.1, pp. 1–51 (cit. on p. 1).

Hewitt, Carl (1990). 'The challenge of open systems'. In: *The foundations of artificial intelligence: a sourcebook*. Cambridge University Press, pp. 383–395 (cit. on p. 4).

Hirayama, Katsutoshi and Makoto Yokoo (1997). 'Distributed Partial Constraint Satisfaction Problem'. In: *International Conference on Principles and Practice of Constraint Programming*. Springer, pp. 222–236 (cit. on p. 27).

Horling, Bryan and Victor Lesser (2005). 'A Survey of Multi-Agent Organizational Paradigms'. In: *The Knowledge Engineering Review* 19.4, pp. 281–316 (cit. on pp. 3, 4).

iChongqing (2020). *The First Fire Drill for High-Rise Fire Fighting Drones Was Held in Dazu, Chongqing*. https://www.youtube.com/watch?v=WFqThcMIN7A. (Visited on 01/11/2021) (cit. on p. 3).

IEEE Robots (2019). *The Colossus Robot*. https://robots.ieee.org/robots/colossus. (Visited on 01/11/2021) (cit. on p. 3).

Imran, Muhammad et al. (2014). 'AIDR: Artificial Intelligence for Disaster Response'. In: *Proceedings of the 23rd International Conference on World Wide Web*, pp. 159–162 (cit. on p. 3).

Jahn, Uwe et al. (2020). 'A Taxonomy for Mobile Robots: Types, Applications, Capabilities, Implementations, Requirements, and Challenges'. In: *Robotics* 9.4, p. 109 (cit. on p. 11).

Jennings, Nicholas R. et al. (2014). 'Human-agent Collectives'. In: *Communications of the ACM* 57.12, pp. 80–88 (cit. on p. 2).

Joslin, David E. and David P. Clements (1999). 'Squeaky Wheel Optimization'. In: *JAIR* 10, pp. 353–373 (cit. on p. 89).

Juárez, Julio, Cipriano Santos and Carlos A. Brizuela (2021). 'A Comprehensive Review and a Taxonomy Proposal of Team Formation Problems'. In: *ACM CSUR* 54.7, pp. 1–33 (cit. on p. 29).

Junges, Robert and Ana L. C. Bazzan (2008). 'Evaluating the Performance of DCOP Algorithms in a Real World, Dynamic Problem'. In: *AAMAS*. Vol. 2, pp. 599–606 (cit. on pp. 22, 60).

Keeney, Ralph L. and Howard Raiffa (1993). *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*. Cambridge University Press (cit. on p. 6).

Kepner, Jeremy (2009). *Parallel MATLAB for Multicore and Multinode Computers*. SIAM (cit. on p. 82).

Khamis, Alaa, Ahmed Hussein and Ahmed Elmogy (2015). 'Multi-robot Task Allocation: A Review of the State-of-the-Art'. In: *Cooperative Robots and Sensor Networks*, pp. 31–51 (cit. on p. 11).

Khan, Md. Mosaddek, Long Tran-Thanh and Nicholas R. Jennings (2018). 'A Generic Domain Pruning Technique for GDL-based DCOP Algorithms in Cooperative Multi-Agent Systems'. In: *AAMAS*, pp. 1595–1603 (cit. on p. 27).

Kho, Johnsen, Alex Rogers and Nicholas R. Jennings (2009). 'Decentralized Control of Adaptive Sampling in Wireless Sensor Networks'. In: *ACM Transations on Sensor Networks* 5.3, pp. 1–35 (cit. on p. 87).

Kiekintveld, Christopher et al. (2010). 'Asynchronous Algorithms for Approximate Distributed Constraint Optimization with Quality Bounds'. In: *AAMAS*, pp. 133–140 (cit. on pp. 24, 88).

Kim, Yoonheui, Michael Krainin and Victor Lesser (2011). 'Effective Variants of the Max-Sum Algorithm for Radar Coordination and Scheduling'. In: *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. Vol. 2, pp. 357–364 (cit. on p. 60).

Kitano, Hiroaki and Satoshi Tadokoro (2001). 'RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems'. In: *AI Magazine* 22.1, pp. 39–53 (cit. on pp. 1, 3, 7, 53).

Kitano, Hiroaki, Satoshi Tadokoro et al. (1999). 'RoboCup Rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research'. In: *International Conference on Systems, Man, and Cybernetics*. Vol. 6. IEEE, pp. 739–743 (cit. on p. 53).

Kleiner, Alexander et al. (2013). 'RMASBench: Benchmarking Dynamic Multi-Agent Coordination in Urban Search and Rescue'. In: *AAMAS*, pp. 1195–1196. URL: https://gitlab.com/lcpz/rmasbench (visited on 01/11/2021) (cit. on p. 53).

Knuth, Donald E. (2005). *The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations*. Pearson Education (cit. on p. 44).

Koes, Mary, Illah Nourbakhsh and Katia Sycara (2005). 'Heterogeneous Multirobot Coordination with Spatial and Temporal Constraints'. In: *AAAI*. Vol. 5, pp. 1292–1297 (cit. on p. 31).

Kolobov, Mausam and Andrey Kolobov (2012). *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers (cit. on p. 14).

Korsah, G. Ayorkor (2011). 'Exploring Bounded Optimal Coordination for Heterogeneous Teams with Cross-Schedule Dependencies'. PhD thesis. Carnegie Mellon University (cit. on pp. 30, 31).

Korsah, G. Ayorkor, Anthony Stentz and M. Bernardine Dias (2013). 'A comprehensive taxonomy for multi-robot task allocation'. In: *International Journal of Robotics Research* 32.12, pp. 1495–1512 (cit. on pp. 11, 30).

Kraus, Sarit, Onn shehory and Gilad Taase (2003). 'Coalition Formation with Uncertain Heterogeneous Information'. In: *AAMAS*, pp. 1–8 (cit. on p. 12).

– (2004). 'The Advantages of Compromising in Coalition Formation with Incomplete Information'. In: *AAMAS*. Vol. 2, pp. 588–595 (cit. on p. 12).

Krausburg, Tabajara, Jürgen Dix and Rafael H. Bordini (2021). 'Feasible Coalition Sequences'. In: *AAMAS*, pp. 719–727 (cit. on pp. 14, 30).

Krizmancic, Marko et al. (2020). 'Cooperative Aerial-Ground Multi-Robot System for Automated Construction Tasks'. In: *IEEE Robotics and Automation Letters* 5.2 (cit. on p. 31).

Kroese, Dirk .P, Thomas Taimre and Zdravko I. Botev (2011). *Handbook of Monte Carlo Methods*. John Wiley & Sons (cit. on p. 26).

Kschischang, Frank R., Brendan J. Frey and Hans-Andrea Loeliger (2001). 'Factor Graphs and the Sum-Product Algorithm'. In: *IEEE Transations on Information Theory* 47.2, pp. 498–519 (cit. on pp. 20, 22, 24).

Kumar, Akshat, Shlomo Zilberstein and Marc Toussaint (2011). 'Scalable Multiagent Planning Using Probabilistic Inference'. In: *AAAI* (cit. on p. 28).

Kurniawati, Hanna (2021). 'Partially Observable Markov Decision Processes (POMDPs) and Robotics'. In: arXiv: `2107.07599 [cs.RO]` (cit. on p. 14).

Lee, Hyun-Rok and Taesik Lee (2021). 'Multi-agent reinforcement learning algorithm to solve a partially-observable multi-agent problem in disaster response'. In: *European Journal of Operational Research* 291.1, pp. 296–308 (cit. on p. 18).

Leeuwen, Cornelis Jan van and Przemyslaw Pawelczak (2017). 'CoCoA: A Non-Iterative Approach to a Local Search (A)DCOP Solver'. In: *AAAI*. Vol. 31. 1 (cit. on pp. 21, 25).

Leite, Allan R. and Fabricio Enembreck (2019a). 'Evaluating Incomplete DCOP Algorithms On Large-Scale Problems'. In: *International Joint Conference on Neural Networks*. IEEE, pp. 1–8 (cit. on p. 60).

– (2019b). 'Using Collective Behavior of Coupled Oscillators for Solving DCOP'. In: *JAIR* 64, pp. 987–1023 (cit. on pp. 21, 25).

Leite, Allan R., Fabricio Enembreck and Jean-Paul A. Barthes (2014). 'Distributed Constraint Optimization Problems: Review and perspectives'. In: *Expert Systems with Applications* 41.11, pp. 5139–5157 (cit. on p. 26).

Lesser, Victor and Daniel Corkill (2014). 'Challenges for Multi-Agent Coordination Theory Based on Empirical Observations'. In: *AAMAS*, pp. 1157–1160 (cit. on p. 26).

Lewis, Richard L., Andrew Howes and Satinder Singh (2014). 'Computational Rationality: Linking Mechanism and Behavior Through Bounded Utility Maximization'. In: *Topics in Cognitive Science* 6.2, pp. 279–311 (cit. on p. 28).

Liu, Chun and Andreas Kroll (2015). 'Memetic algorithms for optimal task allocation in multi-robot systems for inspection problems with cooperative tasks'. In: *Soft Computing* 19.3, pp. 567–584 (cit. on p. 31).

Loeliger, Hans-Andrea (2004). 'An Introduction to Factor Graphs'. In: *IEEE Signal Processing Magazine* 21.1, pp. 28–41 (cit. on p. 20).

London Datastore (2021a). *London Fire Brigade Incident Records*. https://data.london.gov.uk/dataset/london-fire-brigade-incident-records. (Visited on 01/11/2021) (cit. on pp. 7, 60).

– (2021b). *London Fire Brigade Mobilisation Records*. https://data.london.gov.uk/dataset/london-fire-brigade-mobilisation-records. (Visited on 01/11/2021) (cit. on pp. 7, 60).

Luo, Lingzhi, Nilanjan Chakraborty and Katia Sycara (2015). 'Distributed Algorithms for Multirobot Task Assignment with Task Deadline Constraints'. In: *IEEE Transactions on Automation Science and Engineering* 12.3, pp. 876–888 (cit. on p. 29).

Lynch, Nancy A. (1996). *Distributed Algorithms*. Elsevier (cit. on p. 66).

Macarthur, Kathryn S., Alessandro Farinelli et al. (2010). 'Efficient, Superstabilizing Decentralised Optimisation for Dynamic Task Allocation Environments'. In: *International Workshop on Optimization in Multi-Agent systems (OptMAS)*, pp. 25–32 (cit. on pp. 27, 88).

Macarthur, Kathryn S., Ruben Stranders et al. (2011). 'A Distributed Anytime Algorithm for Dynamic Task Allocation in Multi-Agent Systems'. In: *AAAI* (cit. on p. 27).

Mahadevan, Sridhar and Mauro Maggioni (2007). 'Proto-value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes'. In: *Journal of Machine Learning Research* 8, pp. 2169–2231 (cit. on p. 15).

Maheswaran, Rajiv T., Jonathan P. Pearce and Milind Tambe (2004). 'Distributed Algorithms for DCOP: A Graphical-Game-Based Approach'. In: *ISCA PDCS*, pp. 432–439 (cit. on p. 21).

Maheswaran, Rajiv T., Milind Tambe et al. (2004). 'Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling'. In: *AAMAS*. Vol. 1, pp. 310–317 (cit. on p. 60).

Mahmud, Saaduddin, Moumita Choudhury et al. (2020). 'AED: An Anytime Evolutionary DCOP Algorithm'. In: *AAMAS*, pp. 825–833 (cit. on pp. 21, 25).

Mahmud, Saaduddin, Md. Mosaddek Khan and Nicholas R. Jennings (2020). 'On Population-Based Algorithms for Distributed Constraint Optimization Problems'. In: arXiv: 2009.01625 [cs.AI] (cit. on pp. 20, 21, 24–26).

Mailler, Roger, Huimin Zheng and Anton Ridgway (2018). 'Dynamic, distributed constraint solving and thermodynamic theory'. In: *JAAMAS* 32.1, pp. 188–217 (cit. on p. 5).

Maoudj, Abderraouf et al. (2015). 'Multi-agent Approach for Task Allocation and Scheduling in Cooperative Heterogeneous Multi-Robot Team: Simulation Results'. In: *13th International Conference on Industrial Informatics*. IEEE, pp. 179–184 (cit. on p. 30).

Martello, Silvano and Paolo Toth (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc. (cit. on p. 13).

Mataric, Maja J. (1993). 'Designing Emergent Behaviors: From Local Interactions to Collective Intelligence'. In: *International Conference on Simulation of Adaptive Behavior*. MIT Press, pp. 432–441 (cit. on p. 3).

Meisels, Amnon (2007). *Distributed Search by Constrained Agents*. Springer (cit. on pp. 18, 70).

Melo, Francisco S. and Manuela Veloso (2011). 'Decentralized MDPs with sparse interactions'. In: *AIJ* 175.11, pp. 1757–1789 (cit. on p. 16).

Michalak, Tomasz P. et al. (2014). 'Implementation and Computation of a Value for Generalized Characteristic Function Games'. In: *ACM Transactions on Economics and Computation* 2.4, p. 16 (cit. on p. 13).

Miloradović, Branko et al. (2019). 'Tamer: Task Allocation in Multi-robot Systems Through an Entity-Relationship Model'. In: *International Conference on Principles and Practice of Multi-Agent Systems*. Springer, pp. 478–486 (cit. on p. 11).

Modi, Pragnesh Jay et al. (2005). 'Adopt: Asynchronous distributed constraint optimization with quality guarantees'. In: *AIJ* 161.1-2, pp. 149–180 (cit. on p. 22).

Murphy, Robin R. (2014). *Disaster Robotics*. MIT press (cit. on pp. 1, 3).

– (2016). 'Emergency Informatics: Using Computing to Improve Disaster Management'. In: *Computer* 49.5, pp. 19–27 (cit. on pp. 3, 29).

Murphy, Robin R., Satoshi Tadokoro and Alexander Kleiner (2016). 'Disaster Robotics'. In: *Springer Handbook of Robotics*. Springer. Chap. 60, pp. 1577–1604 (cit. on pp. 3, 29).

Myerson, Roger B. (1977). 'Graphs and Cooperation in Games'. In: *Mathematics of operations research* 2.3, pp. 225–229 (cit. on p. 12).

– (1991). *Game Theory: Analysis of Conflict*. Harvard University Press (cit. on p. 6).

Nair, Ranjit, Maayan Roth and Makoto Yohoo (2004). 'Communication for Improving Policy Computation in Distributed POMDPs'. In: *AAMAS*. Vol. 3, pp. 1098–1105 (cit. on p. 28).

Nair, Ranjit, Pradeep Varakantham et al. (2005). 'Networked distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs'. In: *AAAI*. Vol. 5, pp. 133–139 (cit. on p. 28).

Nanjanath, Maitreyi and Maria Gini (2010). 'Repeated auctions for robust task execution by a robot team'. In: *Robotics and Autonomous Systems* 58.7, pp. 900–909 (cit. on p. 30).

Nelke, Sofia Amador, Steven Okamoto and Roie Zivan (2020). 'Market Clearing-based Dynamic Multi-agent Task Allocation'. In: *ACM Transactions on Intelligent Systems and Technology* 11.1, pp. 1–25 (cit. on pp. 4, 30, 31, 33, 60, 88).

Nguyen, Duc Thien et al. (2019). 'Distributed Gibbs: A Linear-Space Sampling-Based DCOP Algorithm'. In: *JAIR* 64, pp. 705–748 (cit. on pp. 23, 88).

Nijs, Frits de et al. (2021). 'Constrained Multiagent Markov Decision Processes: a Taxonomy of Problems and Algorithms'. In: *JAIR* 70, pp. 955–1001 (cit. on p. 14).

Nisan, Noam et al., eds. (2007). *Algorithmic Game Theory*. Cambridge University Press (cit. on p. 12).

Nunes, Ernesto, Marie Manner et al. (2017). 'A taxonomy for task allocation problems with temporal and ordering constraints'. In: *JRAS* 90, pp. 55–70 (cit. on pp. 1, 3, 5, 11, 34, 35, 62, 74, 76).

Nunes, Ernesto, Mitchell McIntire and Maria Gini (2017). 'Decentralized multi-robot allocation of tasks with temporal and precedence constraints'. In: *Advanced Robotics* 31.22, pp. 1193–1207 (cit. on p. 30).

Okamoto, Steven, Roie Zivan, Aviv Nahon et al. (2016). 'Distributed Breakout: Beyond Satisfaction'. In: *IJCAI*, pp. 447–453 (cit. on pp. 21, 24, 88).

Okolloh, Ory (2009). 'Ushahidi, or 'testimony': Web 2.0 tools for crowdsourcing crisis information'. In: *Participatory Learning and Action* 59.1, pp. 65–70 (cit. on p. 2).

Osborne, Martin J. and Ariel Rubinstein (1994). *A Course in Game Theory*. MIT Press (cit. on p. 6).

Papadimitriou, Christos H. (1993). *Computational Complexity*. Pearson (cit. on pp. 4, 36).

Paraskevopoulos, Dimitris C. et al. (2017). 'Resource Constrained Routing and Scheduling: Review and Research Prospects'. In: *European Journal of Operational Research* 263.3, pp. 737–754 (cit. on p. 29).

Parker, Lynne E., Daniela Rus and Gaurav S. Sukhatme (2016). 'Multiple Mobile Robot Systems'. In: *Springer Handbook of Robotics*. Springer, pp. 1335–1384 (cit. on p. 11).

Pearce, Jonathan P. and Milind Tambe (2007). 'Quality Guarantees on k-Optimal Solutions for Distributed Constraint Optimization Problems'. In: *IJCAI*, pp. 1446–1451 (cit. on p. 23).

Peri, Or and Amnon Meisels (2013). 'Synchronizing for Performance-DCOP Algorithms'. In: *ICAART*, pp. 5–14 (cit. on p. 20).

Petcu, Adrian (2007). 'A Class of Algorithms for Distributed Constraint Optimization'. PhD thesis. École polytechnique fédérale de Lausanne (cit. on pp. 5, 19, 26).

Petcu, Adrian and Boi Faltings (2005a). 'A Scalable Method for Multiagent Constraint Optimization'. In: *AAAI* (cit. on pp. 22, 87).

– (2005b). 'Superstabilizing, fault-containing distributed combinatorial optimization'. In: *AAAI*. 20 449 (cit. on pp. 23, 27).

Ponda, Sameera S. et al. (2015). 'Cooperative Mission Planning for Multi-UAV Teams. Handbook of Unmanned Aerial Vehicles'. In: Springer, pp. 1447–1490 (cit. on pp. 11, 15, 88).

Pränatare, Fredrik, Herman Appelgren and Fredrik Heintz (2021). 'Anytime Heuristic and Monte Carlo Methods for Large-Scale Simultaneous Coalition Structure Generation and Assignment'. In: *AAAI* 35.13, pp. 11317–11324 (cit. on p. 30).

Pränatare, Fredrik and Fredrik Heintz (2020). 'An anytime algorithm for optimal simultaneous coalition structure generation and assignment'. In: *JAAMAS* 34.1, pp. 1–31 (cit. on p. 30).

Pujol-Gonzalez, Marc, Jesus Cerquides, Alessandro Farinelli, Pedro Meseguer and Juan A. Rodriguez-Aguilar (2014). 'Binary Max-Sum for Multi-Team Task Allocation in RoboCup Rescue'. In: (cit. on pp. 7, 53).

Pujol-Gonzalez, Marc, Jesus Cerquides, Alessandro Farinelli, Pedro Meseguer and Juan Antonio Rodriguez-Aguilar (2015). 'Efficient Inter-Team Task Allocation in Robocup Rescue'. In: pp. 413–421 (cit. on pp. 53, 54, 59).

Pujol-Gonzalez, Marc, Jesus Cerquides, Pedro Meseguer et al. (2018). 'Decentralized Dynamic Task Allocation for UAVs with Limited Communication Range'. In: arXiv: `1809.07863 [cs.NI]` (cit. on p. 24).

Pynadath, David V. and Milind Tambe (2002). 'The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models'. In: *JAIR* 16, pp. 389–423 (cit. on pp. 15, 28).

Queralta, Jorge Pena et al. (2020). 'Collaborative Multi-Robot Search and Rescue: Planning, Coordination, Perception, and Active Vision'. In: *IEEE Access* 8, pp. 191617–191643 (cit. on p. 29).

Rahwan, Talal, Tomasz Michalak and Nicholas R. Jennings (2012). 'A Hybrid Algorithm for Coalition Structure Generation'. In: *AAAI*. Vol. 26. 1 (cit. on p. 69).

Rahwan, Talal, Tomasz P. Michalak et al. (2015). 'Coalition structure generation: A survey'. In: *AIJ* 229, pp. 139–174 (cit. on pp. 12, 69).

Rahwan, Talal, Tri-Dung Nguyen et al. (2013). 'Coalitional Games via Network Flows'. In: *AAAI*, pp. 324–331 (cit. on p. 13).

Rahwan, Talal, Sarvapali D. Ramchurn et al. (2009). 'An Anytime Algorithm for Optimal Coalition Structure Generation'. In: *JAIR* 34, pp. 521–567 (cit. on pp. 12, 69, 86).

Ramchurn, Sarvapali D., Alessandro Farinelli et al. (2010). 'Decentralized Coordination in RoboCup Rescue'. In: *The Computer Journal* 53.9, pp. 1447–1461 (cit. on pp. 4, 27, 33, 59, 67).

Ramchurn, Sarvapali D., Joel E. Fischer et al. (2015). 'A Study of Human-Agent Collaboration for Multi-UAV Task Allocation in Dynamic Environments'. In: *AAAI* (cit. on p. 24).

Ramchurn, Sarvapali D., Trung Dong Huynh, Yuki Ikuno et al. (2015). 'HAC-ER: A Disaster Response System Based on Human-Agent Collectives'. In: *AAMAS*, pp. 533–541 (cit. on p. 24).

Ramchurn, Sarvapali D., Trung Dong Huynh, Feng Wu et al. (2016). 'A Disaster Response System based on Human-Agent Collectives'. In: *JAIR* 57, pp. 661–708 (cit. on pp. 17, 18, 24).

Ramchurn, Sarvapali D., Maria Polukarov et al. (2010). 'Coalition Formation with Spatial and Temporal Constraints'. In: *AAMAS*, pp. 1181–1188 (cit. on pp. 3, 4, 17, 33, 36–38, 45, 50–52, 60, 63, 78, 87).

Ramchurn, Sarvapali D., Feng Wu et al. (2016). 'Human-Agent Collaboration for Disaster Response'. In: *JAAMAS* 30.1, pp. 82–111 (cit. on pp. 4, 17, 18, 33).

Redding, Joshua D. et al. (2012). 'Scalable, MDP-based planning for multiple, cooperating agents'. In: *American Control Conference*. IEEE, pp. 6011–6016 (cit. on pp. 15, 16).

Rizk, Yara, Mariette Awad and Edward W. Tunstel (2019). 'Cooperative Heterogeneous Multi-Robot Systems: A Survey'. In: *ACM CSUR* 52.2, pp. 1–31 (cit. on p. 29).

RoboCup Rescue Simulator and Agent Development Framework Manual (2021). URL: `https://rescuesim.robocup.org/resources/documentation` (visited on 01/11/2021) (cit. on pp. 53, 55).

Rogers, Alex et al. (2011). 'Bounded approximate decentralised coordination via the max-sum algorithm'. In: *AIJ* 175.2, pp. 730–759 (cit. on p. 24).

Rollon, Emma and Javier Larrosa (2012). 'Improved Bounded Max-Sum for Distributed Constraint Optimization'. In: *International Conference on Principles and Practice of Constraint Programming*. Springer, pp. 624–632 (cit. on p. 24).

Ross, G. Terry and Richard M. Soland (1975). 'A branch and bound algorithm for the generalized assignment problem'. In: *Mathematical Programming* 8.1, pp. 91–103 (cit. on p. 4).

Rossi, Francesca, Peter Van Beek and Toby Walsh, eds. (2006). *Handbook of Constraint Programming*. Elsevier (cit. on pp. 6, 20).

Roth, Maayan, Reid Simmons and Manuela Veloso (2005). 'Decentralized Communication Strategies for Coordinated Multi-Agent Policies'. In: *Multi-Robot Systems. From Swarms to Intelligent Automata*. Vol. 3. Springer, pp. 93–105 (cit. on p. 28).

Roy, Nicholas et al. (2021). 'From Machine Learning to Robotics: Challenges and Opportunities for Embodied Intelligence'. In: arXiv: 2110.15245 [cs.RO] (cit. on p. 14).

Sandholm, Tuomas W., Kate Larson et al. (1999). 'Coalition Structure Generation with Worst Case Guarantees'. In: *AIJ* 111.1-2, pp. 209–238 (cit. on pp. 11–13, 28, 84).

Sandholm, Tuomas W. and Victor R. T. Lesser (1997). 'Coalitions among computationally bounded agents'. In: *AIJ* 94.1-2, pp. 99–137 (cit. on p. 13).

Scerri, Paul et al. (2005). 'Allocating Tasks in Extreme Teams'. In: *AAMAS*, pp. 727–734 (cit. on p. 29).

Schneider, Marco (1993). 'Self-stabilization'. In: *ACM CSUR* 25.1, pp. 45–67 (cit. on p. 26).

Schwarzrock, Janaína et al. (2018). 'Solving task allocation problem in multi Unmanned Aerial Vehicles systems using Swarm intelligence'. In: *Engineering Applications of Artificial Intelligence* 72, pp. 10–20 (cit. on p. 31).

Sedgewick, Robert (1977). 'Permutation Generation Methods'. In: *ACM CSUR* 9.2, pp. 137–164 (cit. on pp. 82, 83).

Seenu, N. et al. (2020). 'Review on state-of-the-art dynamic task allocation strategies for multiple-robot systems'. In: *Industrial Robot: the international journal of robotics research and application* 47.6, pp. 929–942 (cit. on p. 29).

Service, Travis C. and Julie A. Adams (2011). 'Coalition formation for task allocation: Theory and algorithms'. In: *JAAMAS* 22.2, pp. 225–248 (cit. on pp. 29, 89).

Seuken, Sven and Shlomo Zilberstein (2008). 'Formal models and algorithms for decentralized decision making under uncertainty'. In: *JAAMAS* 17.2, pp. 190–250 (cit. on pp. 14–16, 28).

Shehory, Onn and Sarit Kraus (1998). 'Methods for task allocation via agent coalition formation'. In: *AIJ* 101.1-2, pp. 165–200 (cit. on pp. 3, 11, 12, 29).

Shoham, Yoav and Kevin Leyton-Brown (2008). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press (cit. on p. 1).

Silver, David and Joel Veness (2010). 'Monte-Carlo Planning in Large POMDPs'. In: Neural Information Processing Systems (cit. on p. 17).

Singh, Arambam J., Poulami Dalapati and Animesh Dutta (2014). 'Multi Agent Based Dynamic Task Allocation'. In: *Agent and Multi-Agent Systems: Technologies and Applications*. Springer, pp. 171–182 (cit. on p. 29).

Skaltsis, George Marios, Hyo-Sang Shin and Antonios Tsourdos (2021). 'A survey of task allocation techniques in MAS'. In: *International Conference on Unmanned Aircraft Systems*. IEEE, pp. 488–497 (cit. on p. 11).

Smith, Andrew J. et al. (2019). 'Real-time distributed non-myopic task selection for heterogeneous robotic teams'. In: *Autonomous Robots* 43.3, pp. 789–811 (cit. on p. 16).

Smith, Reid G. and Randall Davis (1981). 'Frameworks for Cooperation in Distributed Problem Solving'. In: *IEEE Transactions on Systems, Man, and Cybernetics* 11.1, pp. 61–70 (cit. on p. 14).

Spaan, Matthijs T. J., Tiago S. Veiga and Pedro U. Lima (2015). 'Decision-theoretic planning under uncertainty with information rewards for active cooperative perception'. In: *JAAMAS* 29.6, pp. 1157–1185 (cit. on p. 16).

Stankovic, John A. et al. (2013). *Deadline scheduling for real-time systems: EDF and related algorithms*. Vol. 460. Reprint of the original 1998 edition. Springer Science & Business Media (cit. on pp. 69, 80, 84).

Stone, Peter, Richard S. Sutton and Gregory Kuhlmann (2005). 'Reinforcement Learning for RoboCup Soccer Keepaway'. In: *Adaptive Behavior* 13.3, pp. 165–188 (cit. on p. 14).

Stranders, Ruben, Francesco Maria Delle Fave et al. (2010). 'A Decentralised Coordination Algorithm for Mobile Sensors'. In: *AAAI* (cit. on p. 24).

Stranders, Ruben, Alessandro Farinelli et al. (2009). 'Decentralised Coordination of Mobile Sensors Using the Max-Sum Algorithm'. In: *AAAI* (cit. on p. 24).

Su, Xing et al. (2018). 'Two innovative coalition formation models for dynamic task allocation in disaster rescues'. In: *Journal of Systems Science and Systems Engineering* 27.2, pp. 215–230 (cit. on p. 29).

Sultanik, Evan A., Robert N. Lass and William C. Regli (2009). 'Dynamic Configuration of Agent Organizations'. In: *AAAI*, pp. 305–311 (cit. on p. 27).

Suslova, Elina and Pooyan Fazli (2020). 'Multi-Robot Task Allocation with Time Window and Ordering Constraints'. In: *IROS*. IEEE, pp. 6909–6916 (cit. on p. 30).

Sutton, Richard S. (1996). 'Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding'. In: *Advances in Neural Information Processing Systems*, pp. 1038–1044 (cit. on p. 15).

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Second edition. The MIT Press (cit. on p. 14).

Svensson, Kim et al. (2013). 'Solving the Coalition Structure Generation Problem on a GPU'. In: *International Workshop on Optimisation in Multi-Agent Systems* (cit. on p. 89).

Tae, Hyunchul and Byung-In Kim (2015). 'A Branch-and-Price Approach for the Team Orienteering Problem with Time Windows'. In: *International Journal of Industrial Engineering* 22.2 (cit. on p. 89).

Tarapore, Danesh, Roderich Groß and Klaus-Peter Zauner (2020). 'Sparse Robot Swarms: Moving Swarms to Real-World Applications'. In: *Frontiers in Robotics and AI* 7, p. 83 (cit. on p. 72).

Taylor, Matthew E., Manish Jain, Yanquin Jin et al. (2010). 'When Should There be a "Me" in "Team"?: Distributed Multi-Agent Optimization Under Uncertainty'. In: *AAMAS*. Vol. 1, pp. 109–116 (cit. on p. 88).

Taylor, Matthew E., Manish Jain, Prateek Tandon et al. (2011). 'Distributed on-line multi-agent optimization under uncertainty: Balancing exploration and exploitation'. In: *Advances in Complex Systems* 14.03, pp. 471–528 (cit. on p. 88).

Tkach, Itshak and Sofia Amador (2021). 'Towards addressing dynamic multi-agent task allocation in law enforcement'. In: *JAAMAS* 35.1, pp. 1–18 (cit. on pp. 4, 30, 31, 33, 60, 88).

Torreño, Alejandro et al. (2017). 'Cooperative Multi-Agent Planning: A Survey'. In: *ACM CSUR* 50.6, pp. 1–32 (cit. on pp. 11, 14).

Tsimenidis, Stefanos (2020). 'Limitations of Deep Neural Networks: a discussion of G. Marcus' critical appraisal of deep learning'. In: arXiv: `2012.15754 [cs.AI]` (cit. on p. 14).

Ueda, Suguru et al. (2010). 'Coalition Structure Generation Based on Distributed Constraint Optimization'. In: *AAAI*. Vol. 24. 1 (cit. on p. 89).

Van Steen, Maarten and Andrew S. Tanenbaum (2017). *Distributed Systems*. Third edition. CreateSpace Independent Publishing Platform (cit. on pp. 4, 5).

Vansteenwegen, Pieter and Aldy Gunawan (2019). *Orienteering Problems: Models and Algorithms for Vehicle Routing Problems with Profits*. Switzerland: Springer Nature (cit. on pp. 7, 8, 63, 77, 78, 89).

Vasudevan, Rama K. et al. (2021). 'Off-the-shelf deep learning is not enough, and requires parsimony, Bayesianity, and causality'. In: *npj Computational Materials* 7.1, pp. 1–6 (cit. on p. 14).

Verma, Janardan Kumar and Virender Ranga (2021). 'Multi-Robot Coordination Analysis, Taxonomy, Challenges and Future Scope'. In: *Journal of Intelligent & Robotic Systems* 102.1, pp. 1–36 (cit. on p. 11).

Vieira, Renata et al. (2007). 'On the formal semantics of speech-act based communication in an agent-oriented programming language'. In: *JAIR* 29, pp. 221–267 (cit. on p. 65).

Vig, Lovekesh and Julie A. Adams (2006). 'Multi-Robot Coalition Formation'. In: *IEEE Transactions on Robotics* 22.4, pp. 637–649 (cit. on p. 29).

– (2007). 'Coalition Formation: From Software Agents to Robots'. In: *Journal of Intelligent and Robotic Systems* 50.1, pp. 85–118 (cit. on p. 29).

Vinyals, Meritxell et al. (2011). 'Quality Guarantees for Region Optimal DCOP Algorithms'. In: *AAMAS*. Vol. 1, pp. 133–140 (cit. on p. 24).

Whitbrook, Amanda, Qinggang Meng and Paul W. H. Chung (2015). 'A Novel Distributed Scheduling Algorithm for Time-Critical Multi-Agent Systems'. In: *IROS*. IEEE, pp. 6451–6458 (cit. on p. 30).

– (2017). 'Reliable, Distributed Scheduling and Rescheduling for Time-Critical, Multiagent Systems'. In: *IEEE Transactions on Automation Science and Engineering* 15.2, pp. 732–747 (cit. on p. 30).

– (2019). 'Addressing robustness in time-critical, distributed, task allocation algorithms'. In: *Applied Intelligence* 49.1, pp. 1–15 (cit. on p. 30).

Wolsey, Laurence A. (2020). *Integer Programming*. Second edition. John Wiley & Sons (cit. on pp. 7, 33, 60, 61).

Wu, Feng, Sarvapali D. Ramchurn and Xiaoping Chen (2016). 'Coordinating Human-UAV Teams in Disaster Response'. In: *IJCAI*, pp. 524–530 (cit. on pp. 17, 18).

Yan, Eugene (2021). *The First Rule of Machine Learning: Start without Machine Learning*. `https://eugeneyan.com/writing/first-rule-of-ml`. (Visited on 01/11/2021) (cit. on p. 14).

Yang, Tianpei et al. (2021). 'Exploration in Deep Reinforcement Learning: A Comprehensive Survey'. In: arXiv: `2109.06668 [cs.AI]` (cit. on p. 14).

Ye, Dayong, Minjie Zhang and Danny Sutanto (2013). 'Self-Adaptation-Based Dynamic Coalition Formation in a Distributed Agent Network: A Mechanism and a Brief Survey'. In: *IEEE Transactions on Parallel and Distributed Systems* 24.5, pp. 1042–1051 (cit. on p. 31).

Ye, Dayong, Minjie Zhang and Athanasios V. Vasilakos (2016). 'A Survey of Self-Organization Mechanisms in Multiagent Systems'. In: *IEEE Transactions on Systems, Man, and Cybernetics* 47.3, pp. 441–461 (cit. on p. 11).

Yedidsion, Harel, Roie Zivan and Alessandro Farinelli (2018). 'Applying max-sum to teams of mobile sensing agents'. In: *Engineering Applications of Artificial Intelligence* 71, pp. 87–99 (cit. on p. 24).

Yeoh, William (2010). 'Speeding Up Distributed Constraint Optimization Search Algorithms'. PhD thesis. University of Southern California (cit. on p. 20).

Yeoh, William et al. (2015). 'Incremental DCOP Search Algorithms for Solving Dynamic DCOP Problems'. In: *IEEE International Conference on Web Intelligence and Intelligent Agent Technology*. Vol. 2, pp. 257–264 (cit. on p. 27).

Yokoo, Makoto, Edmund H. Durfee et al. (1998). 'The Distributed Constraint Satisfaction Problem: Formalization and Algorithms'. In: *IEEE Transactions on Knowledge and Data Engineering* 10.5, pp. 673–685 (cit. on p. 18).

Yokoo, Makoto and Katsutoshi Hirayama (1996). 'Distributed Breakout Algorithm for Solving Distributed Constraint Satisfaction Problems'. In: *AAMAS*. MIT Press Cambridge, pp. 401–408 (cit. on p. 24).

Yokoo, Makoto, Toru Ishida et al. (1992). 'Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving'. In: *International Conference on Distributed Computing Systems*. IEEE, pp. 614–621 (cit. on p. 65).

Zaoad, Syeed Abrar et al. (2021). 'Accelerating Message Passing Operation of GDL-Based Constraint Optimization Algorithms Using Multiprocessing'. In: *IEEE ISPA 2021*. IEEE (cit. on p. 27).

Zeng, Dajun and Katia Sycara (1996). 'How can an agent learn to negotiate?' In: *International Workshop on Agent Theories, Architectures, and Languages*. Springer, pp. 233–244 (cit. on p. 28).

Zhang, Weixiong et al. (2005). 'Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks'. In: *AIJ* 161.1-2, pp. 55–87 (cit. on pp. 7, 23, 70).

Zhou, Jing et al. (2020). 'Task Allocation for Multi-agent Systems Based on Distributed Many-objective Evolutionary Algorithm and Greedy Algorithm'. In: *IEEE Access* (cit. on p. 31).

Zick, Yair, Georgios Chalkiadakis and Edith Elkind (2012). 'Overlapping Coalition Formation Games: Charting the Tractability Frontier'. In: *AAMAS*. Vol. 2, pp. 787–794 (cit. on p. 13).

Zilberstein, Shlomo (1996). 'Using Anytime Algorithms in Intelligent Systems'. In: *AI Magazine* 17.3, pp. 73–83 (cit. on p. 4).

Zivan, Roie (2021). 'Applying Multi-Agent Optimization to Realistic Scenarios, including IOT Applications'. OptLearnMAS at AAMAS 2021. URL: https://optlearnmas21.github.io/#roie (visited on 01/11/2021) (cit. on p. 88).

Zivan, Roie, Steven Okamoto and Hilla Peled (2014). 'Explorative Anytime Local Search for Distributed Constraint Optimization'. In: *AIJ* 212, pp. 1–26 (cit. on pp. 7, 23, 24, 26, 67, 68, 71).

Zivan, Roie and Hilla Peled (2012). 'Max/Min-sum Distributed Constraint Optimization through Value Propagation on an Alternating DAG'. In: *AAMAS*. Vol. 1, pp. 265–272 (cit. on p. 24).

Zivan, Roie, Harel Yedidsion et al. (2015). 'Distributed constraint optimization for teams of mobile sensing agents'. In: *JAAMAS* 29.3, pp. 495–536 (cit. on p. 26).

Zlot, Robert Michael (2006). 'An Auction-Based Approach to Complex Task Allocation for Multirobot Teams'. PhD thesis. The Robotics Institute, Carnegie Mellon University (cit. on p. 29).

Zou, Hui and Yan Xi (2021). 'Decentralised Task Allocation Using GDL Negotiations in Multi-agent System'. In: *Cognitive Robotics (in press)*. DOI: 10.1016/j.cogr.2021.07.003 (cit. on p. 6).