Multi-Robot Coordination Under Temporal Uncertainty



Charlie Street

Pembroke College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2022

Acknowledgements

I'm really grateful to have had such a wonderful group of people supporting me during this DPhil. First, I want to thank my supervisors Nick and Bruno. In addition to your amazing academic support, your constant reassurance helped me greatly in dealing with those times I got a tad stressed. I'd also like to thank Manuel for your support from Germany, and HRI-EU more generally for funding my DPhil. I'm sorry I was never able to visit HRI-EU (pesky pandemic), but maybe there's still a chance in the future? Thanks also to Jon and Chris for examining this thesis and engaging with it in the way that you did. I also want to thank everyone in the GOALS group, and the ORI more widely, for being a joy to work with and be around on a day-to-day basis.

Outside of the ORI, I've lived in the same (very pink) house for the last four years with some great people. To Ella, Ed, and Hugh, thank you for being so kind and welcoming upon my nervous arrival to Oxford, and for dealing with my nonsense for three years, particularly during the pandemic. Thanks also to Giac and Nic for bearing with me for the last year while I finished this DPhil. Special thanks go to Poison for being a very good cat.

Outside of Oxford, I'd like to thank my friends for a) not forgetting about me, and b) making my non-work life much more enjoyable. In particular, I want to thank Joe, Mel, and Tom for making my Monday evenings during the pandemic (and beyond) incredibly entertaining with very questionable games. Back home in Weymouth, thanks to Cameron and Matt for the quizzes, Jackbox sessions, and support in my eight year long quest to learn how to install a printer.

I couldn't have got through this DPhil, or anything ever in fact, without the loving support of my family. Thank you to Mum, Dad, and Jenna for always welcoming me back to Weymouth and encouraging me in whatever I do. Finally, thank you to Harri. The last four years have been tough, what with the living apart in a pandemic thing, but you've helped me so much, and reminded me (correctly) on many occasions that everything was going to be OK. Finishing this DPhil also marks the end of that living apart thing, so onwards to Birmingham!

Abstract

Sources of *temporal uncertainty* affect the duration and start time of robot actions during execution. For example, mobile robots may slip on uneven terrain, slowing them down. The presence of multiple robots in the environment contributes towards temporal uncertainty, as robot interactions such as *congestion* affect navigation performance. Existing multi-robot coordination solutions often disregard temporal uncertainty through simplifying assumptions such as fixed, identical action durations, which simplifies the problem at the cost of inefficient execution-time behaviour. To synthesise multi-robot behaviour that is robust to unexpected temporal disturbances, we must explicitly capture temporal uncertainty during coordination.

In this thesis, we present techniques for effective multi-robot coordination under temporal uncertainty. To represent temporal uncertainty, we construct probabilistic models of action duration for each spatiotemporal situation an action may be executed in. With this, we build formal multi-robot models which accurately capture asynchronous robot execution in continuous time. We then develop planning and task allocation solutions which reason over the effects of temporal uncertainty to synthesise efficient multi-robot behaviour. Further, we consider the problem of policy evaluation, i.e. evaluating properties of synthesised multi-robot behaviour prior to execution, and present two approaches which trade between the accuracy of the model and the solution methods used for policy evaluation. Empirically, our methods outperform solutions which ignore temporal uncertainty, use simplified models, or constrain robot behaviour to reduce temporal uncertainty.

Contents

List of Figures viii

List of Tables xii

List of Abbreviations xiv

List of Notation xv

1 Introduction 1
- 1.1 Overview . . . 1
- 1.2 Contributions . . . 4
- 1.3 Structure of Thesis . . . 6
- 1.4 Publications . . . 7

2 Background 8
- 2.1 Topological Maps . . . 9
- 2.2 Markov Decision Processes (MDPs) . . . 9
  - 2.2.1 Multi-Agent MDPs (MMDPs) . . . 12
- 2.3 Continuous-Time Markov Chains (CTMCs) . . . 13
  - 2.3.1 Transient Analysis . . . 14
  - 2.3.2 Reachability Probabilities . . . 15
  - 2.3.3 Expected Time to Reach an Absorbing State . . . 16
- 2.4 Phase-Type Distributions (PTDs) . . . 17
  - 2.4.1 Classes of PTDs . . . 18
  - 2.4.2 Fitting PTDs from Empirical Data . . . 19
  - 2.4.3 Approximating Deterministic Delays . . . 20
- 2.5 Markov Automata (MA) . . . 20
- 2.6 Poisson Binomial Distributions . . . 22

3 Related Work 24
- 3.1 Multi-Robot Planning and Modelling Under Uncertainty . . . 25
  - 3.1.1 Multi-Agent Path Finding (MAPF) . . . 26
  - 3.1.2 Discrete-Time Planning Under Uncertainty . . . 28

3.1.3 Discrete-Time Multi-Robot Planning under Uncertainty . . . 30
3.1.4 Multi-Robot Planning on Single-Robot Models 31
3.1.5 Planning Under Robot Interactions 32
3.1.6 Continuous-Time Planning Under Uncertainty 34
3.1.7 Planning in Non-Stationary Environments 35
3.1.8 Multi-Robot Planning in Partially Observable Environments 36
3.1.9 Multi-Robot Planning Under Resource Constraints 39
3.1.10 Planning versus Reinforcement Learning 40
3.2 Policy Evaluation for Multi-Robot Systems 41
3.2.1 Discrete Event Simulators (DESs) 41
3.2.2 Model Checking Multi-Robot Systems 42
3.2.3 Combining Planning with Model Checking 43
3.3 Multi-Robot Task Allocation (MRTA) 44
3.3.1 Classical MRTA Solutions 44
3.3.2 Auctioning Solutions for MRTA 45
3.3.3 Alternative MRTA Approaches 47
3.3.4 MRTA Under Uncertainty 48
3.3.5 Simultaneous Task Allocation and Planning (STAP) 49

4 Phenomena that Affect Robot Navigation Durations 51
4.1 Introduction 51
4.2 The Effects of Congestion 53
4.2.1 Map Resource Groups 56
4.3 Contexts 58
4.3.1 Describing the Context for Multi-Robot Systems 59
4.3.2 Interaction Bands 61
4.4 Constructing Duration Models from Empirical Data 62
4.5 Summary 64

5 Context-Aware Modelling for Multi-Robot Systems Under Uncertainty 66
5.1 Introduction 67
5.2 Context-Aware Multi-Robot MA 69
5.2.1 MRS Assumptions 69
5.2.2 Single-Robot MA 71
5.2.3 Multi-Robot MA 74
5.3 Context-Aware Multi-Robot Simulation 75
5.3.1 Robot Policies 76
5.3.2 The CAMAS Algorithm 78
5.4 Complexity Analysis 80

5.4.1 MRMA Space Complexity 80
5.4.2 CAMAS Time Complexity 81
5.5 Experiments 81
5.5.1 Experimental Setup 81
5.5.2 Evaluating Duration-Based Metrics 85
5.5.3 CAMAS Simulation Speed 89
5.5.4 Evaluating Failure-Based Metrics 89
5.6 Discussion 93
5.7 Summary 94

6 Congestion-Aware Policy Synthesis for Multi-Robot Systems 96
6.1 Introduction 97
6.2 Time-Varying Markov Automata (TVMA) 99
6.3 Problem Formulation 100
6.4 Framework 101
6.5 Modelling the Evolution of Congestion 103
6.5.1 Representing Congestion 103
6.5.2 Computing Congestion Probabilities 104
6.5.3 Defining a PRT 106
6.6 Planning Under Congestion 106
6.6.1 Single-Robot Planning Models 107
6.6.2 LRTDP for TVMAs 110
6.6.3 Constructing a Robot's Route CTMC 114
6.6.4 Complexity Analysis 116
6.7 Policy Execution Model Refinement 117
6.8 Experiments 120
6.8.1 Experimental Setup 120
6.8.2 Validation in Synthetic Environments 122
6.8.3 Validation in a Simulated Environment 132
6.9 Discussion 136
6.10 Summary 137

7 Proactive Multi-Robot Task Allocation Under Spatiotemporal Uncertainty 139
7.1 Introduction 140
7.2 Problem Formulation 143
7.2.1 Environment Model 143
7.2.2 Tasks and Schedules 144
7.2.3 Task Announcement Model 145
7.2.4 MRTA Under Spatiotemporal Uncertainty 148

7.3 Proactive SSI Auctioning Under Spatiotemporal Uncertainty 149
7.3.1 Auctioning Framework 150
7.3.2 Computing Bids 153
7.3.3 IWPs 156
7.3.4 Complexity Analysis 157
7.4 Experiments 159
7.4.1 Experimental Setup 159
7.4.2 Measuring Performance as Robots Increase 162
7.4.3 Measuring Performance as Tasks Increase 166
7.4.4 Schedule Cost Scalability 167
7.5 Discussion 168
7.6 Summary 169

8 Conclusion **171**
8.1 Summary of Contributions 171
8.2 Future Work 175

References **177**

Appendices

A The Floyd-Warshall Algorithm **199**

B Labelled Real-Time Dynamic Programming (LRTDP) **201**

C Fitting PTDs using Moment Matching **205**

D Fitting PTDs using Maximum Likelihood Estimation **207**

E Sequential Single-Item Auctioning (SSI) **210**

List of Figures

1.1 Example multi-robot domains. 2

2.1 An example topological map. Nodes are red, and edges are blue. In this example, edges are bidirectional, i.e. if v_i and v_j are connected, then $\{(v_i, v_j), (v_j, v_i)\} \subseteq E$. 10
2.2 An example MDP $\mathcal{M}$, where $\bar{s}_{\mathcal{M}} = s_1$, $\delta_{\mathcal{M}}(s_1, a_1, s_2) = 0.7$, $\delta_{\mathcal{M}}(s_1, a_1, s_3) = 0.3$, $\delta_{\mathcal{M}}(s_2, a_2, s_1) = 0.5$, $\delta_{\mathcal{M}}(s_2, a_2, s_3) = 0.5$, and $\delta_{\mathcal{M}}(s_3, a_3, s_3) = 1$. 11
2.3 An example two robot MMDP $\mathcal{M}$, where a_i^j refers to robot r_i's jth action. In $\mathcal{M}$, $\bar{s}_{\mathcal{M}} = s_1$, $\delta_{\mathcal{M}}(s_1, (a_1^1, a_2^2), s_1) = 0.2$, $\delta_{\mathcal{M}}(s_1, (a_1^1, a_2^2), s_2) = 0.8$, $\delta_{\mathcal{M}}(s_1, (a_1^1, a_2^3), s_1) = 0.1$, $\delta_{\mathcal{M}}(s_1, (a_1^1, a_2^3), s_3) = 0.9$, $\delta_{\mathcal{M}}(s_2, (a_1^2, a_2^4), s_3) = 1$, $\delta_{\mathcal{M}}(s_3, (a_1^3, a_2^3), s_2) = 0.7$, and $\delta_{\mathcal{M}}(s_3, (a_1^3, a_2^3), s_3) = 0.3$. 12
2.4 An example of a CTMC, and the corresponding uniformised CTMC. Solid arrows are immediate transitions, and dashed arrows are exponential transitions. 14
2.5 Examples of different PTD structures. Dashed arrows are exponential transitions, and concentric circles are absorbing states. 19
2.6 An example MA $\mathcal{A}$, where $\bar{s}_{\mathcal{A}} = s_1$, $\delta_{\mathcal{A}}(s_1, a_1, s_2) = 0.7$, $\delta_{\mathcal{A}}(s_1, a_1, s_3) = 0.3$, $\delta_{\mathcal{A}}(s_4, a_2, s_3) = 0.45$, $\delta_{\mathcal{A}}(s_4, a_2, s_4) = 0.55$, $\Delta_{\mathcal{A}}(s_2, s_1) = \lambda_1$, $\Delta_{\mathcal{A}}(s_2, s_3) = \lambda_3$, $\Delta_{\mathcal{A}}(s_2, s_4) = \lambda_2$, and $\Delta_{\mathcal{A}}(s_3, s_3) = \lambda_4$. Solid arrows are immediate transitions, and dashed arrows are exponential transitions. 21

3.1 The core components required for multi-robot coordination, and the relationships between them. 25
3.2 The spectrum of auctioning solutions. 45

4.1 Duration data for a simulated robot navigating on a topological edge, which demonstrates temporal uncertainty. Data was collected using the techniques in Section 4.4. 52
4.2 A demonstration of how congestion affects navigation duration. The circles with arrows show robots in the map, and coloured lines are possible robot paths. 54

4.3 An example demonstrating spatial dependencies between map resources. Blue circles are topological nodes, and white lines are edges. 57
4.4 An MRS acting on a topological map, where squares represent robots. Robot r_1 is about to navigate along edge (v_1, v_3); robots r_2 and r_3 are navigating along (v_3, v_1) and (v_2, v_3) respectively. 58
4.5 The relationships between congestion, contexts, and interaction bands. 61
4.6 An example edge partition, where edges with the same colour are of the same length. 63
4.7 Example PTDs for different scalar contexts on a topological edge, where the PTDs are fit from empirical duration data. 63

5.1 An MRS on a topological map, where squares represent robots and there is a door at v_2. Robot r_1 is stationary and about to navigate along topological edge (v_1, v_2). 71
5.2 A simulation run through a 2 robot MRMA $\mathcal{A}_J$. Solid arrows represent immediate transitions, dashed arrows represent exponential transitions, and concentric circles represent PTD absorbing states. . 76
5.3 The warehouse environment with topological map overlaid. Blue circles show initial robot locations. 82
5.4 The supermarket environment with topological map overlaid. Blue circles show initial robot locations. 82
5.5 Experimental map resource groups for Chapter 5. Nodes and edges highlighted in red are included in the map resource group. 83
5.6 The sum of task durations for each method. 86
5.7 The empirical CDFs for each method across two tasks in the warehouse. 88
5.8 CAMAS scalability results. 89

6.1 An overview of the congestion-aware planning framework. Robots plan sequentially. Each robot r_i queries the PRT to build its TVMA $\mathcal{A}_i^{\mathrm{TV}}$, which is solved to obtain a policy π_i. A route CTMC $\mathcal{Q}_i$ representing this policy is inserted into the PRT. After planning, all CTMCs can be updated iteratively to evaluate the behaviour induced by the synthesised policies, if required. 102
6.2 The pipeline for computing the congestion probabilities on edge e at time t using the PRT. Note, there may not be a CTMC for all robots at the time of use. 104
6.3 An example TVMA for robot r_i at time t. Solid arrows are immediate transitions, dashed arrows are exponential transitions, and concentric circles represent PTD absorbing states. Note that in Fig. 6.3b, the immediate transition probabilities are dependent on t. 108

6.4 The process of generating a route CTMC from a robot's policy. Solid arrows are immediate/MDP transitions, dashed arrows are exponential transitions, and concentric circles represent PTD absorbing states. 115
6.5 The process of collapsing immediate transitions into exponential transitions. Solid arrows are immediate transitions; dashed arrows are exponential transitions. 116
6.6 The maps used for the synthetic experiments. 122
6.7 The simulated makespans for each of the 3 planning methods in the 5×5 warehouse map. 123
6.8 Planner scalability across the warehouse maps. Fig. 6.8a-6.8c show the scalability on the 5×5 warehouse. Fig. 6.8d-6.8f show the scalability on the 15×15 warehouse, and Fig. 6.8g-6.8i show the scalability on the warehouse with tunnel map. 125
6.9 The effect the number of interaction bands have on planning time. . 127
6.10 The effect the number of interaction bands have on makespan. . . . 127
6.11 The number of Alg. 6.2 iterations for each heuristic to reach convergence. 129
6.12 The difference in expected time error between the initial and final CTMCs. Each point gives the mean error across the entire team. . . 131
6.13 The difference in time-bounded reachability error between the initial and final CTMCs. Each point gives the mean error across the entire team. 131
6.14 A congested warehouse environment with a topological map overlaid. The circles with arrows show robots in the map. 132
6.15 The observed makespans for each of the 3 methods in the Stage simulation. 133
6.16 The time taken for each robot to reach its goal in simulation. Green circles show the expected time computed on the single-robot CTMCs produced by Alg. 6.2. Blue circles show the expected time computed on the MRMA using CAMAS. The problem number indicates how many robots travelled through a tunnel in Fig. 6.14. 135

7.1 Robots operating alongside fruit pickers in a fruit field. 141
7.2 A task CTMC $\mathcal{Q}_\tau$ and the corresponding location distribution P^L_τ when at different states in the model. Concentric circles represent absorbing states, λ values are exponential rates, and the value above each absorbing state is the corresponding PTL, defined by the labelling function. 146

7.3 A task CTMC $\mathcal{Q}_\tau$ with a star topology, where concentric circles are absorbing states, λ values are exponential rates, and above each absorbing state is the corresponding PTL, as defined by the labelling function. 147
7.4 A framework that extends SSI for proactive task allocation. Red blocks are standard SSI; green blocks are our extensions. 150
7.5 An example demonstrating how robots proactively execute tasks. Starting from node v at time 0, robot r_1 navigates to WP v_w and waits for task τ to be announced. Upon announcement, r_1 navigates to the task location and services τ, where the PTLs are v_1, v_2, and v_3. Arrows between different nodes represent multiple topological edges. The coloured values show the terms in Eq. 7.6 and the lower bound we use in practice. 151
7.6 An example topological map, where edge weights are the expected duration of navigating that edge. Robot r_1 is initially located at node v_0. 157
7.7 The performance of each method as we increase the team size. . . . 162
7.8 The performance of each method as we increase the number of tasks. 165
7.9 Scalability of the schedule cost SC. 167

List of Tables

5.1 The error between Gazebo and each method, where each value is the sum over the KS test results for each task. Bold values give the lowest error for that problem setup. All values are unitless. 87

5.2 Evaluation of the average tasks completed metric. In the first table, bold values have the closest mean to Gazebo, and highlighted cells show methods whose distribution is not statistically significantly different to Gazebo. In the second table, highlighted cells show methods whose distribution is not statistically significantly different to the method with the closest mean to Gazebo (written beneath the environment). Statistical significance is determined using a two-sided Mann-Whitney U test, where $p = 0.05$. 90

5.3 Evaluation of the successful robots metric, following the same convention as Table 5.2. 91

5.4 Evaluation of the successful run metric, following the same convention as Table 5.2. 92

6.1 The mean makespans for the results in Fig. 6.7. Bold values have the lowest mean for that number of robots. Methods with highlighted cells do not produce statistically significantly higher makespans than the method with lowest mean, according to a one-sided Mann-Whitney U test with $p = 0.05$. 124

6.2 The mean times in seconds $\pm$ standard deviation for each of the 3 heuristics to complete Alg. 6.2. Bold values have the lowest mean for that number of robots. 130

6.3 The p-values from a one-sided Mann-Whitney U test for the makespans in Fig. 6.15. Bold values are those where the congestion-aware planner produces statistically significantly lower makespans than the relevant baseline, where $p = 0.05$ 134

7.1 The mean waiting durations for the results in Fig. 7.7. The problem column is written <environment>-<CTMC structure>-<number of robots>, where Sup denotes the supermarket, and Grid denotes the 10×10 grid world. Bold values have the lowest mean for that problem. Methods with highlighted cells do not produce statistically significantly higher waiting durations than the method with lowest mean, according to a one-sided Mann-Whitney U test with $p = 0.05$. 163

7.2 The mean waiting durations for the results in Fig. 7.8. The table is structured as in Table 7.1, except for the number in the problem column, which now refers to the number of tasks. 166

List of Abbreviations

CAMAS . . . Context-aware multi-agent simulator.

CTMC . . . . Continuous-time Markov chain.

DES . . . . . Discrete-event simulator.

DTMC . . . . Discrete-time Markov chain.

IWP . . . . . Intermediate waiting point.

LRTDP . . . . Labelled real-time dynamic programming.

LRTDP-TVMA LRTDP for TVMAs.

MA . . . . . Markov automata/automaton.

MAPF . . . . . Multi-agent path finding.

MDP . . . . . Markov decision process.

MMDP . . . . Multi-agent MDP.

MRMA . . . . Multi-robot MA.

MRS . . . . . Multi-robot system.

MRTA . . . . . Multi-robot task allocation.

PRT . . . . . Probabilistic reservation table.

PTD . . . . . Phase-type distribution.

PTL . . . . . Potential task location.

SSI . . . . . Sequential single-item auctioning.

SSP . . . . . Stochastic shortest-path problem.

TVMA . . . . Time-varying MA.

TVMA-SSP . SSP over TVMAs.

WP . . . . . Waiting point.

List of Notation

Robots

R . . . . . A set of robots.

r_i . . . . . The ith robot.

Markov Models

A . . . . . A finite set of actions.

$\mathcal{A}$ . . . . . A Markov automaton.

AP . . . . . A finite set of atomic propositions.

$\mathcal{A}^{\mathrm{TV}}$ . . . . . A time-varying Markov automaton.

Δ . . . . . An exponential transition function.

δ . . . . . An immediate transition function.

$init$ . . . . . An initial state distribution.

Lab . . . . . A state labelling function.

$\mathcal{M}$ . . . . . A Markov decision process.

$\mathcal{P}$ . . . . . A phase-type distribution.

$\mathbb{P}$ . . . . . The set of phase-type distributions.

$\mathcal{Q}$ . . . . . A continuous-time Markov chain.

S . . . . . A finite set of states.

s^f . . . . . The absorbing state of a phase-type distribution.

Context-Aware Topological Maps

C . . . . . The complete set of contexts.

C_x . . . . . The set of contexts observable at map resource $x \in X$.

c_i . . . . . A function which returns the context for robot r_i at a given map resource at the instant r_i triggers an action.

$\texttt{cnt}_i$ . . . . . A function which returns the number of other robots at a given map resource at the instant robot r_i triggers an action.

E . . . . . The set of topological edges.

g . . . . . A map resource group function.

I . . . . . The complete set of interaction bands.

I_x . . . . . The set of interaction bands at map resource $x \in X$.

ι_x^j . . . . . The jth interaction band at map resource $x \in X$.

ρ_E . . . . . A context-dependent edge duration distribution function.

$\texttt{src}$ . . . . . A function which returns the source of a topological edge.

$\mathcal{T}_C$ . . . . . A context-aware topological map.

$\texttt{trg}$ . . . . . A function which returns the target of a topological edge.

V . . . . . The set of topological nodes.

X . . . . . The set of map resources (nodes and edges).

Probabilistic Reservation Table

$cong$ . . . . . A function which returns the congestion probabilities for robot r_i on topological edge $e \in E$ at time t.

$update$ . . . . A function which returns the update function for the probabilistic reservation table.

Proactive Multi-Robot Task Allocation

B_τ^Φ . . . . . The service time distribution for task $\tau \in T$ under allocation Φ.

$\texttt{IWP}_i^j$ . . . . . A function which returns the intermediate waiting point for the jth task in robot r_i's schedule.

Ω_i^j . . . . . A function which returns the expected sum of task service times starting from the jth task in robot r_i's schedule.

P_τ^A . . . . . A distribution over the time until announcement for task $\tau \in T$.

P_τ^L . . . . . A distribution over the announcement location for task $\tau \in T$.

PTL . . . . . The set of potential task location-task pairs.

Φ . . . . . An allocation which maps from robots to schedules.

ϕ_i . . . . . The schedule for robot r_i.

SC . . . . . The schedule cost function.

T . . . . . A finite set of tasks.

τ . . . . . A task.

W_τ^Φ . . . . . The waiting duration distribution for task $\tau \in T$ under allocation Φ.

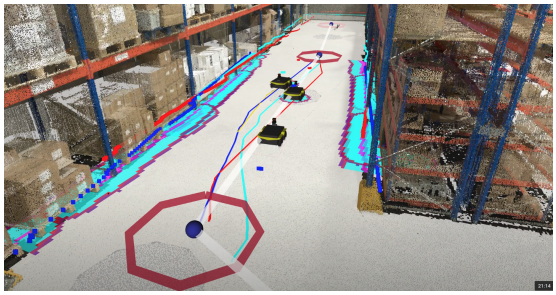
1 Introduction

Contents

1.1 Overview **1**
1.2 Contributions **4**
1.3 Structure of Thesis **6**
1.4 Publications **7**

1.1 Overview

The demand for multi-robot systems (MRSs) is increasing, due to their performance, flexibility, and fault tolerance [1, 2]. Successful multi-robot deployments have been completed in fulfilment centres (e.g. Fig. 1.1a) [3], in fruit fields (e.g. Fig. 1.1b) [4], and on roads [5]. In each of these environments, robot behaviour is affected by the stochastic dynamics of the environment and the other robots. For example, a mobile robot operating in an office may fail to navigate upon a door being closed unexpectedly. Alternatively, a robot may be unable to dock at a charging station if another robot is charging for longer than expected. To effectively deploy MRSs in dynamic and uncertain environments, we require coordination solutions that are robust to these sources of uncertainty.



(a) Clearpath Jackals navigating in a warehouse.



(b) Thorvald agricultural robots. Reproduced from [6].

Figure 1.1: Example multi-robot domains.

Multi-robot coordination is achieved through *planning* and *task allocation* methods, which are often decoupled for scalability. Planning techniques synthesise robot behaviour given a *formal model* of the system. The success of planning is limited by model accuracy, which indicates the capacity to capture and predict execution-time robot behaviour. Multi-robot planning solutions extend single-robot techniques to consider dependencies between robots sharing the same physical space. Multi-robot task allocation (MRTA) methods assign tasks to robots to achieve a global goal [7]. Robots plan for the tasks they are assigned. Given accurate planning models, we can apply *policy evaluation* techniques, such as simulation or model checking, to evaluate task-level properties of synthesised multi-robot behaviour prior to execution, such as the probability of completing a task within a time bound. Policy evaluation can support coordination through execution monitoring [8], or optimising the number of robots [9] etc. In this thesis, we tackle multi-robot modelling, planning, task allocation, and policy evaluation.

Robot coordination under uncertainty is a well studied field. Traditionally, robot uncertainty is captured over discrete action outcomes [10] or the current state [11]. Stochastic outcomes occur due to navigation failure [12], battery depletion [13], or stochastic features of the environment such as hazards [14], resources [15], and doors [16]. However, less attention has been paid to *temporal uncertainty*, which affects the duration and start times of robot actions.

Temporal uncertainty occurs in almost any robot environment, where action durations are affected by environmental disturbances, such as unknown obstacles or

adverse weather conditions, or the robot itself. For example, a mobile robot's tire may slip on a carpet while navigating through an office, slowing it down. Further, robots may have to wait for stochastic processes in the environment, such as order arrival in a fulfilment centre, before beginning task execution. In MRSs, the presence of multiple robots contributes towards temporal uncertainty. Robot interactions such as *congestion* increase uncertainty over action duration. Congestion occurs when multiple mobile robots operate in the same physical space simultaneously. To resolve congestion, robots must manoeuvre to avoid each other, where these manoeuvres are often highly dependent on the precise spatial and temporal situation, and thus are hard to predict accurately at planning time. Moreover, robots may have to wait stochastically for each other before action execution, such as for collaborative handovers. To synthesise multi-robot behaviour that is robust to temporal disturbances caused by the robots and environment, we must consider temporal uncertainty explicitly during coordination.

Existing multi-robot coordination techniques often disregard temporal uncertainty through simplifying assumptions such as all actions having the same fixed duration [17, 18]. Those that do consider temporal uncertainty often rely on simplistic duration models, such as exponential [19, 20], Gaussian [6], or Poisson models [21]. This fails to capture the complex temporal dynamics that govern robot action durations and other external processes which affect the MRS, such as task announcement. This causes our expectations of behaviour during planning to diverge from what we observe during execution, where action durations are continuous and stochastic. This divergence results in inefficient multi-robot behaviour, and so for efficient coordination we require less restrictive duration models.

The effects of robot interactions are often ignored during planning, which causes robots to experience significant delays during execution. Alternatively, robot interactions are prevented through synchronised execution, where robots execute actions in lockstep [17], and robots are forbidden from entering the same location simultaneously [18]. This simplifies coordination at the cost of suboptimal execution-time behaviour, as robots have to wait for each other to finish acting

before a subsequent action can be performed [22]. To accurately capture execution-time multi-robot behaviour, we must explicitly consider how robot interactions affect stochastic action durations.

In this thesis, we propose approaches that bridge the gap between planning and execution time, synthesising policies that are robust to temporal disturbances caused by robot interactions and other environmental features. However, this produces multiple sources of complexity. First, robot action durations are inherently continuous and stochastic, and so we must model the continuous-time evolution of the system. Second, robot interactions are not realised until execution, and so we must predict them at planning time.

1.2 Contributions

In this thesis, we address continuous-time multi-robot coordination under temporal uncertainty by tackling the following questions:

Q1 - Interactions: How do robot interactions affect action durations?

Q2 - Modelling: Can we build probabilistic models that accurately capture temporal uncertainty and robot interactions?

Q3 - Planning: Can we use these models to construct effective planning mechanisms?

Q4 - Policy Evaluation: Can we apply policy evaluation techniques on these models to evaluate task-level properties of execution-time multi-robot behaviour?

Q5 - Task Allocation: Can we allocate tasks efficiently under spatiotemporal uncertainty?

In answering these questions, we demonstrate that modelling temporal uncertainty explicitly during planning improves the performance and predictability of the MRS. We present novel techniques for modelling asynchronous, continuous-time multi-robot execution, and scalable coordination methods that reason over temporal

uncertainty and robot interactions. We demonstrate the efficacy of these techniques through empirical evaluation in synthetic and simulated domains. Specifically, we contribute the following:

- Techniques for constructing probabilistic duration models under different interaction scenarios, and for collecting robot navigation data to fit these models (**Q1 - Interactions**, **Q2 - Modelling**).

- Multi-robot Markov automata (MRMA), a joint model for a team of robots interacting in a shared space, which allows for asynchronous execution, duration uncertainty, and robot interactions (**Q2 - Modelling**).

- CAMAS, a discrete-event simulator (DES) that samples through an MRMA given a set of robot policies to evaluate task-level metrics of multi-robot behaviour (**Q4 - Policy Evaluation**).

- A planning framework for multi-robot coordination under congestion, where a probabilistic reservation table (PRT) models the effects of congestion on robot navigation, and single-robot planning problems are formulated as time-varying Markov automata (TVMA), a novel formulation for continuous-time planning problems under time-varying dynamics (**Q2 - Modelling**, **Q3 - Planning**).

- An approach to compute probabilistic guarantees over robot behaviour that iteratively refines continuous-time Markov chains (CTMCs) describing robot policy execution to more accurately represent the influence of other robots (**Q2 - Modelling**, **Q4 - Policy Evaluation**).

- A framework which uses CTMC models of stochastic task announcement to extend sequential single-item (SSI) auctioning for proactive task allocation under spatiotemporal uncertainty (**Q2 - Modelling**, **Q5 - Task Allocation**).

1.3 Structure of Thesis

In Chapter 2, we introduce the foundational models and distributions used throughout this thesis, and in Chapter 3, we discuss the relevant literature. To support coordination, in Chapter 4 we define the notion of *contexts* to describe the spatiotemporal situation actions are executed in, and construct probabilistic duration models for each context. We explore how robot interactions affect navigation durations, and focus on congestion, a mode of interaction for mobile robots which occurs when multiple robots share the same location simultaneously. To capture the effects of robot interactions, we fit context-dependent duration distributions from navigation duration data collected in a robot simulator. This chapter combines work published in [23–25].

In Chapter 5, we use context-dependent duration distributions to construct an MRMA which captures asynchronous multi-robot execution in continuous time. The MRMA handles interactions such as congestion directly by observing robots in the joint MRMA state, and selecting appropriate duration distributions upon robot action execution. To validate the MRMA, we present a DES called CAMAS (context-aware multi-agent simulator). CAMAS samples through the MRMA under a set of robot policies to evaluate task-level properties of multi-robot behaviour, such as the time to complete a set of tasks. This work first appeared in [25], presented at AAMAS 2022.

The MRMA scales exponentially in the number of robots [17], and is intractable to solve even for small MRSs. To mitigate this, in Chapter 6 we present a sequential planning framework, where robots plan using single-robot models which capture team behaviour through distributions over congestion. We use a PRT to store continuous-time representations of robot policy execution, and compute the congestion distributions. Moreover, we demonstrate that the models in the PRT can be used to approximate local performance guarantees through standard model checking techniques [26]. This work first appeared in [23], presented at AAMAS 2020, and was later expanded in [24], presented in the IEEE Transactions in Robotics.

The planner in Chapter 6 is limited to local robot objectives, and sources of temporal uncertainty that directly affect robot action durations. Therefore, in Chapter 7 we apply MRTA techniques to coordinate under global objectives. We consider tasks with stochastic announcements, both in space and in time, which we model using CTMCs which can be learned from empirical data. Using these models, we present a proactive MRTA framework which extends SSI auctioning to minimise the expected total waiting time, i.e. the time between task announcement and a robot beginning to service the task. This work first appeared in [27], which is currently under review. Chapter 8 concludes the thesis, and in the Appendix we describe established algorithms that support our contributions.

1.4 Publications

The work in this thesis has been published as the following:

- Charlie Street, Bruno Lacerda, Manuel Mühlig, and Nick Hawes. "Multi-Robot Planning Under Uncertainty with Congestion-Aware Models". In: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2020.

- Charlie Street, Sebastian Pütz, Manuel Mühlig, Nick Hawes, and Bruno Lacerda. "Congestion-Aware Policy Synthesis for Multirobot Systems". In: *IEEE Transactions on Robotics* (2021).

- Charlie Street, Bruno Lacerda, Michal Staniaszek, Manuel Mühlig, and Nick Hawes. "Context-Aware Modelling for Multi-Robot Systems Under Uncertainty". In: *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2022.

- Charlie Street, Bruno Lacerda, Manuel Mühlig, and Nick Hawes. "Right Place, Right Time: Proactive Multi-Robot Task Allocation Under Spatiotemporal Uncertainty". In: *Under Review*. 2022.

2
Background

Contents

2.1 Topological Maps 9
2.2 Markov Decision Processes (MDPs) 9
2.2.1 Multi-Agent MDPs (MMDPs) 12
2.3 Continuous-Time Markov Chains (CTMCs) 13
2.3.1 Transient Analysis 14
2.3.2 Reachability Probabilities 15
2.3.3 Expected Time to Reach an Absorbing State 16
2.4 Phase-Type Distributions (PTDs) 17
2.4.1 Classes of PTDs 18
2.4.2 Fitting PTDs from Empirical Data 19
2.4.3 Approximating Deterministic Delays 20
2.5 Markov Automata (MA) 20
2.6 Poisson Binomial Distributions 22

In this chapter, we introduce environment representations, probabilistic models, and solution methods used throughout this thesis.

Notational Remarks. We write $|X|$ to denote the cardinality of a set X, or the size of vector X. We use $\text{Dist}(X)$ to denote the set of distributions over set X, and $\mathbb{E}[P]$ to denote the expected value of distribution P. For a 2D matrix $\boldsymbol{M}$, $\boldsymbol{M}(i,j)$ is the element at index (i,j); the same applies for vectors. We denote the discrete interval $\{i, \cdots, j\},\ j \geq i$ as $[\![i,j]\!]$.

2.1 Topological Maps

In this thesis, we consider the task-level coordination of MRSs under temporal uncertainty. To support coordination, we represent robot environments as a *topological map*. Topological maps simplify the environment by considering the relevant locations for tasks and navigation decisions, which robots navigate between. With this, we plan over the topological map, and rely on efficient motion planning solutions for navigation and action execution [28–30].

Definition 2.1. A *topological map* is a tuple $\mathcal{T} = \langle V, E \rangle$ where V is a finite set of nodes representing locations in the environment, and $E \subseteq V \times V$ is a set of directed edges which robots can travel on.

We refer to nodes and edges as *map resources*, the set of which is given by $X = E \cup V$. Further, for edge $e = (v, v') \in E$, we use $\texttt{src}(e) = v$ and $\texttt{trg}(e) = v'$ to denote the source and target of that edge, respectively. Robot environments can also be represented in terms of the continuous configuration space of the robots. To synthesise task-level behaviour in configuration space, we must also reason over robot motion. Task and motion planning techniques can synthesise optimal task and motion plans, but are often computationally expensive [31]. By planning on topological maps we decouple task and motion planning, which sacrifices optimality but allows for more scalable coordination. Topological maps allow us to manage the complexity of synthesising multi-robot behaviour under complex models of temporal uncertainty, such as the continuous duration distributions introduced in Chapter 4. All topological maps in this thesis are created manually, however maps can be learned from overhead images [32] or point clouds [33]. An example topological map is shown in Fig. 2.1.

2.2 Markov Decision Processes (MDPs)

Sources of uncertainty are found in almost any environment robots are deployed in. Therefore, it is important to incorporate environmental uncertainty into formal

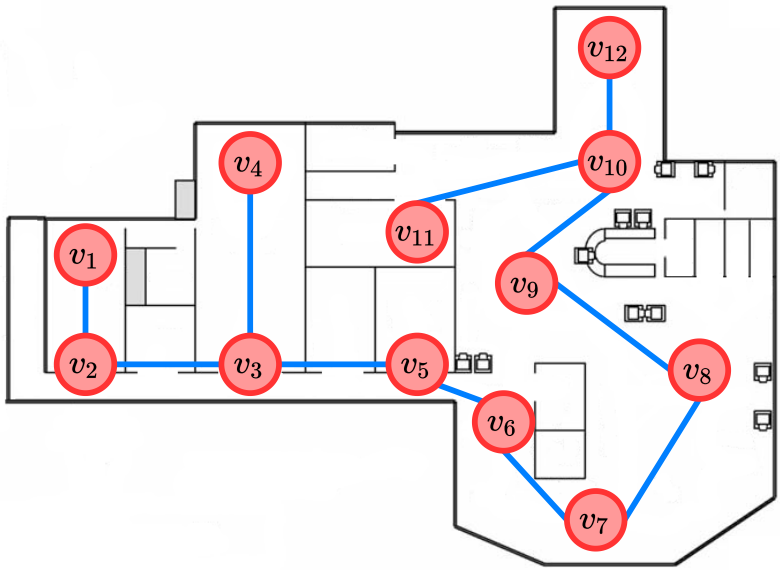


Figure 2.1: An example topological map. Nodes are red, and edges are blue. In this example, edges are bidirectional, i.e. if v_i and v_j are connected, then $\{(v_i, v_j), (v_j, v_i)\} \subseteq E$.

models of robot behaviour. MDPs model systems with non-deterministic action choices and stochastic action outcomes [10]. MDPs are a common formalism for robotics, and have been used to model mobile service robots [34], safe robot exploration [35], robotic manipulation [36] etc. In Chapter 6, we solve stochastic shortest path problems (SSPs) which capture an approximate version of the multi-robot planning problem under congestion, addressing **Q3 - Planning**. SSPs have an indefinite horizon, i.e. a finite but unknown horizon, and can be encoded as an MDP [37].

Definition 2.2. An *SSP MDP* [38] is a tuple $\mathcal{M} = \langle S, \bar{s}, A, \delta, c, G \rangle$, where S is a finite set of states, $\bar{s} \in S$ is the initial state, A is a finite set of actions, and $\delta : S \times A \times S \to [0, 1]$ is a transition function, such that $\delta(s, a, s')$ returns the probability of transitioning to state s' after executing action a in state s. Further, $c : S \times A \to \mathbb{R}_{>0}$ is a cost function, where $c(s, a)$ denotes the cost of executing action a in state s, and $G \subseteq S$ is a set of goal states to be reached.

Throughout this thesis, we use subscripts to denote an element belonging to a model, e.g. $S_{\mathcal{M}}$ and $A_{\mathcal{M}}$ for the state space and action set of MDP $\mathcal{M}$ respectively. The MDPs we build are used for mobile robot navigation planning, where an

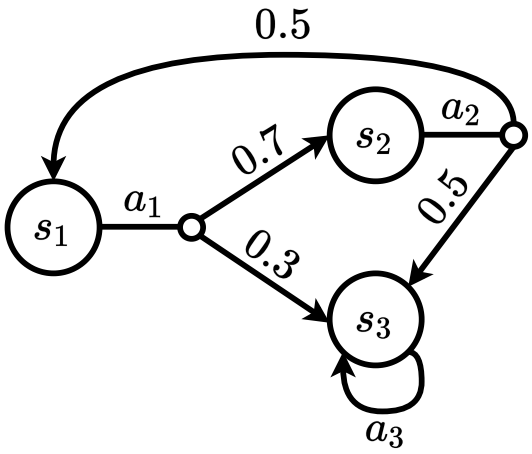


Figure 2.2: An example MDP $\mathcal{M}$, where $\bar{s}_{\mathcal{M}} = s_1$, $\delta_{\mathcal{M}}(s_1, a_1, s_2) = 0.7$, $\delta_{\mathcal{M}}(s_1, a_1, s_3) = 0.3$, $\delta_{\mathcal{M}}(s_2, a_2, s_1) = 0.5$, $\delta_{\mathcal{M}}(s_2, a_2, s_3) = 0.5$, and $\delta_{\mathcal{M}}(s_3, a_3, s_3) = 1$.

MDP state typically includes the robot's location, and actions represent choices of (nearby) locations to move towards. An example MDP is shown in Fig. 2.2. Next, we define the goal of an SSP.

Problem 2.1. Given an SSP MDP $\mathcal{M}$, find a policy $\pi^* : S_{\mathcal{M}} \to A_{\mathcal{M}}$ which minimises the expected cumulative cost to reach a state in $G_{\mathcal{M}}$ from the initial state $\bar{s}_{\mathcal{M}}$, i.e.:

$$\pi^* = \operatorname*{argmin}_{\pi} V_\pi(\bar{s}_{\mathcal{M}}), \tag{2.1}$$

where $V_\pi : S_{\mathcal{M}} \to \mathbb{R}_{\geq 0}$ is a value function which defines the expected cumulative cost to reach a state in $G_{\mathcal{M}}$ from each state s under policy π. Formally:

$$V_\pi(s) = \begin{cases} 0 & s \in G_{\mathcal{M}} \\ c_{\mathcal{M}}(s, \pi(s)) + \sum_{s' \in S_{\mathcal{M}}} \delta_{\mathcal{M}}(s, a, s') \cdot V_\pi(s') & \text{otherwise.} \end{cases} \tag{2.2}$$

Policies define behaviour over an MDP by informing an agent which actions to execute. Policies may be stochastic, and/or defined over state/action histories. However, for SSPs there exists an optimal policy that is deterministic and memoryless [38].

Definition 2.3. A deterministic, memoryless *policy* is a function $\pi : S \to A$, where $\pi(s)$ denotes the action to be executed in state s.

A policy π induces a discrete-time Markov chain (DTMC) $\mathcal{M}^\pi$ over an MDP $\mathcal{M}$ by resolving non-determinism, i.e. only the policy actions are available. SSPs

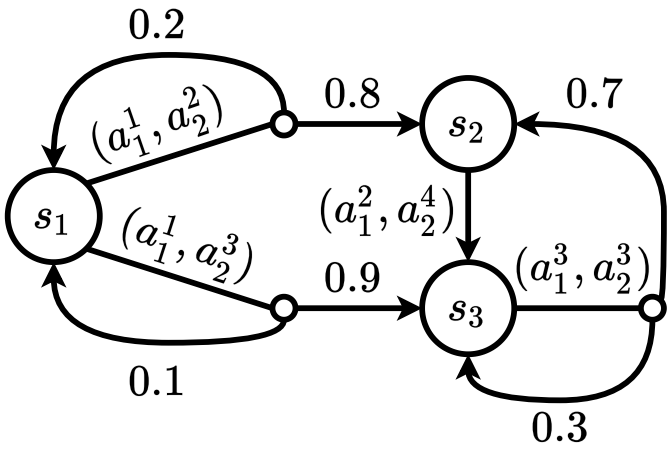


Figure 2.3: An example two robot MMDP $\mathcal{M}$, where a_i^j refers to robot r_i's jth action. In $\mathcal{M}$, $\bar{s}_\mathcal{M} = s_1$, $\delta_\mathcal{M}(s_1, (a_1^1, a_2^2), s_1) = 0.2$, $\delta_\mathcal{M}(s_1, (a_1^1, a_2^2), s_2) = 0.8$, $\delta_\mathcal{M}(s_1, (a_1^1, a_2^3), s_1) = 0.1$, $\delta_\mathcal{M}(s_1, (a_1^1, a_2^3), s_3) = 0.9$, $\delta_\mathcal{M}(s_2, (a_1^2, a_2^4), s_3) = 1$, $\delta_\mathcal{M}(s_3, (a_1^3, a_2^3), s_2) = 0.7$, and $\delta_\mathcal{M}(s_3, (a_1^3, a_2^3), s_3) = 0.3$.

require the existence of at least one proper policy, i.e. a policy that, starting in $\bar{s}$, reaches a goal state with probability 1. Further, for any improper policy, the value of all states under that policy must be ∞, which can be enforced by setting the value of any dead end states to ∞. The infinite dead end values are then propagated back to all other states by Eq. 2.2. Moreover, all goal states must be *absorbing*, i.e. there must be no outgoing transitions from goal states.

2.2.1 Multi-Agent MDPs (MMDPs)

MMDPs are a natural extension of MDPs to multiple agents, where the agents act in a *joint* state and action space [17]. An example MMDP is shown in Fig. 2.3.

Definition 2.4. An *SSP MMDP* for a team of n robots $R = \{r_1, ..., r_n\}$ is an SSP MDP $\mathcal{M} = \langle S, \bar{s}, A, \delta, c, G \rangle$, where the action set is the Cartesian product of individual robot actions, $A = A_1 \times ... \times A_n$, where A_i is the set of actions for robot r_i; and $c : S \times A \to \mathbb{R}_{>0}$ is a cost function defined over joint states and actions.

MMDP states are often factored into local states for each robot $S_{l,i}, \forall i \in [\![1, n]\!]$, and a shared set of global state features S_g, $S = S_{l,1} \times ... \times S_{l,n} \times S_g$. In MMDPs, a joint action is taken at each discrete time step, i.e. each robot executes an action. Action execution is synchronous, i.e. robots execute in lockstep, and all actions

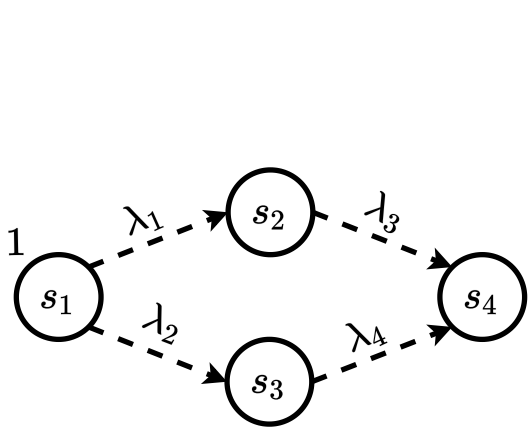
have the same duration. The number of MMDP states increases exponentially in the number of robots, making optimal planning methods intractable. In Chapter 3, we discuss techniques to overcome MMDP scalability and synchronous execution.

2.3 Continuous-Time Markov Chains (CTMCs)

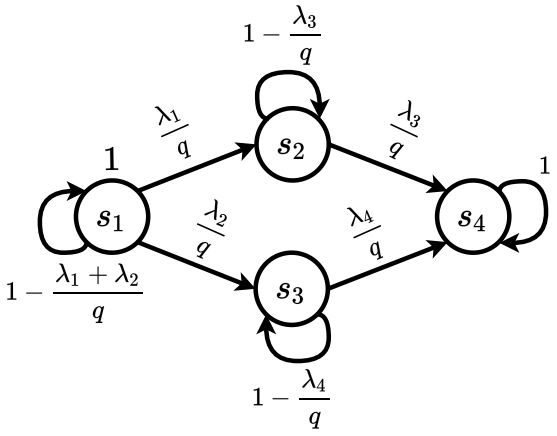
Markov chains are established models for stochastic processes. DTMCs have been used to model environmental hazards [14], resource constraints [15], and disease spread [39]. However, many real world processes operate in continuous time, which we can represent as CTMCs. CTMCs describe the continuous-time evolution of a system as a sequence of exponential distributions [40], and have been used to model networks [41, 42], traffic [43], chemical reaction networks [44] and hospital stays [45]. In this thesis, we use labelled CTMCs to model the continuous-time execution of robot policies (see Chapter 6), and continuous-time task announcement (see Chapter 7). We provide an example CTMC in Fig. 2.4a.

Definition 2.5. A *labelled CTMC* is a tuple $\mathcal{Q} = \langle S, init, \Delta, AP, Lab \rangle$, where S is a finite set of states, and $init : S \to [0, 1]$ gives the probability of a state being the initial state. $\Delta : S \times S \to \mathbb{R}_{>0}$ is an exponential transition function, such that $\Delta(s, s')$ returns the rate between states s and s'. AP is a finite set of atomic propositions, and $Lab : S \to 2^{AP}$ is a labelling function that maps states to atomic propositions that hold in that state.

Exponential transitions are associated with a value $\Delta(s, s')$, which is the rate parameter of an exponential distribution associated with their duration. The probability of the transition firing within time t is given by $1 - e^{-\Delta(s,s') \cdot t}$. The *exit rate* is the sum of outgoing rates in a state s, i.e. $E(s) = \sum_{s' \in S} \Delta(s, s')$. The probability of leaving state s within time t is $1 - e^{-E(s) \cdot t}$; the probability of branching to state s' from s is $\Delta(s, s')/E(s)$. When a state has multiple exponential transitions, a *race condition* occurs, i.e. there is a race over which transition fires. Race conditions can be efficiently resolved by sampling a duration using the exit rate, and sampling a successor state under the branching distribution.



(a) An example CTMC $\mathcal{Q}$, where $init_{\mathcal{Q}}(s_1) = 1$, $\Delta_{\mathcal{Q}}(s_1, s_2) = \lambda_1$, $\Delta_{\mathcal{Q}}(s_1, s_3) = \lambda_2$, $\Delta_{\mathcal{Q}}(s_2, s_4) = \lambda_3$, and $\Delta_{\mathcal{Q}}(s_3, s_4) = \lambda_4$.



(b) The uniformised DTMC $\mathcal{U}_{\mathcal{Q}}$ of CTMC $\mathcal{Q}$ in Fig. 2.4a, where the uniformisation rate is q.

Figure 2.4: An example of a CTMC, and the corresponding uniformised CTMC. Solid arrows are immediate transitions, and dashed arrows are exponential transitions.

This is equivalent to sampling from each exponential transition and selecting the minimum, but is more sample efficient.

2.3.1 Transient Analysis

In Chapter 6, we predict future robot locations using the transient probabilities of a CTMC, i.e. the probability of being in state s at time t. Transient probabilities can be computed as a matrix exponential, but may cause numerical errors [40, 46]. Instead, we normalise a CTMC's exponential transitions through *uniformisation*, and analyse the transient properties of the resulting uniformised DTMC.

Definition 2.6. Let $q \geq \max_{s \in S_{\mathcal{Q}}} E(s)$ be the *uniformisation rate*. The *uniformised DTMC* of a CTMC $\mathcal{Q}$ is a tuple $\mathcal{U}_{\mathcal{Q}} = \langle S_{\mathcal{U}_{\mathcal{Q}}}, init_{\mathcal{U}_{\mathcal{Q}}}, \delta_{\mathcal{U}_{\mathcal{Q}}}, AP_{\mathcal{U}_{\mathcal{Q}}}, Lab_{\mathcal{U}_{\mathcal{Q}}} \rangle$, where $S_{\mathcal{U}_{\mathcal{Q}}}$, $init_{\mathcal{U}_{\mathcal{Q}}}$, $AP_{\mathcal{U}_{\mathcal{Q}}}$, and $Lab_{\mathcal{U}_{\mathcal{Q}}}$ are as in $\mathcal{Q}$, and transition function $\delta_{\mathcal{U}_{\mathcal{Q}}}$ normalises the CTMC transitions against the uniformisation rate q:

$$\delta_{\mathcal{U}_{\mathcal{Q}}}(s, s') = \begin{cases} \frac{\Delta_{\mathcal{Q}}(s,s')}{q} & \text{if } s \neq s' \\ 1 - \frac{\sum_{s'' \neq s} \Delta_{\mathcal{Q}}(s,s'')}{q} & \text{if } s = s'. \end{cases} \tag{2.3}$$

The uniformisation rate q is chosen as the exit rate of the state with minimum expected sojourn time. The sojourn time at each state in $\mathcal{U}_{\mathcal{Q}}$ is exponentially

distributed with rate q. We add self loops such that all states have exit rate q, and the same expected sojourn time. Fig. 2.4b shows the uniformised DTMC for the CTMC in Fig. 2.4a.

For the transient probabilities, we compute a matrix $\mathbf{\Gamma}^t_{\mathcal{U}_\mathcal{Q}}$, where $\mathbf{\Gamma}^t_{\mathcal{U}_\mathcal{Q}}(s, s')$ is the probability of residing in state s' at time t, having started in state s. $\mathbf{\Gamma}^t_{\mathcal{U}_\mathcal{Q}}$ is computed as an infinite sum, where $\boldsymbol{P}_{\mathcal{U}_\mathcal{Q}}$ is the transition matrix of $\mathcal{U}_\mathcal{Q}$, i.e. $\boldsymbol{P}_{\mathcal{U}_\mathcal{Q}}(s, s') = \delta_{\mathcal{U}_\mathcal{Q}}(s, s')$:

$$\mathbf{\Gamma}^t_{\mathcal{U}_\mathcal{Q}} = \sum_{i=0}^{\infty} \left(e^{-q \cdot t} \cdot \frac{(q \cdot t)^i}{i!} \right) \cdot (\boldsymbol{P}_{\mathcal{U}_\mathcal{Q}})^i. \tag{2.4}$$

The first term of the summation in Eq. 2.4 is the Poisson probability that i transitions occur in time t, given the exit rate at each state is q. The second term is the probability of moving between each pair of states in i transitions [40]. In practice, we truncate the infinite sum while retaining a level of precision [47], and rearrange it for efficient computation [40]. We compute the probability of being in state s at time t, denoted $\Pr(s@t)$, as the weighted sum of $\mathbf{\Gamma}^t_{\mathcal{U}_\mathcal{Q}}$ with the initial state distribution:

$$\Pr(s@t) = \sum_{s' \in S_{\mathcal{U}_\mathcal{Q}}} init_{\mathcal{U}_\mathcal{Q}}(s') \cdot \mathbf{\Gamma}^t_{\mathcal{U}_\mathcal{Q}}(s', s). \tag{2.5}$$

2.3.2 Reachability Probabilities

In Chapter 7, we compute discrete task location distributions using the probability that we reach a CTMC state, irrespective of time. We compute reachability probabilities on the *embedded DTMC* of a CTMC, which captures the CTMC branching distributions. The embedded DTMC is abstract of time, unlike the uniformised DTMC.

Definition 2.7. The *embedded DTMC* [40] of a CTMC $\mathcal{Q}$ is a tuple $\mathcal{D}_\mathcal{Q} = \langle S_{\mathcal{D}_\mathcal{Q}}, init_{\mathcal{D}_\mathcal{Q}}, \delta_{\mathcal{D}_\mathcal{Q}}, AP_{\mathcal{D}_\mathcal{Q}}, Lab_{\mathcal{D}_\mathcal{Q}} \rangle$, where $S_{\mathcal{D}_\mathcal{Q}}$, $init_{\mathcal{D}_\mathcal{Q}}$, $AP_{\mathcal{D}_\mathcal{Q}}$ and $Lab_{\mathcal{D}_\mathcal{Q}}$ are as in $\mathcal{Q}$, and transition function $\delta_{\mathcal{D}_\mathcal{Q}}$ is defined such that for states s and s', $\delta_{\mathcal{D}_\mathcal{Q}}(s, s')$ is the probability of transitioning to s' from s:

$$\delta_{\mathcal{D}_\mathcal{Q}}(s,s') = \begin{cases} \frac{\Delta_\mathcal{Q}(s,s')}{E(s)} & \text{if } E(s) \neq 0 \\ 1 & \text{if } E(s) = 0 \text{ and } s = s' \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

For the reachability of state s', we first compute the set of states from which s' is unreachable. Propagating backwards from s' according to $\delta_{\mathcal{D}_\mathcal{Q}}$, we find the states with a non-zero probability of reaching s'; the unreachable set is the remainder of the state space. Next, we consider the probability of reaching s' from state s, which we compute recursively by propagating reachability probabilities through the embedded DTMC [40]:

$$\Pr^{reach}(s,s') = \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{if } s' \text{ not reachable from } s \\ \sum_{s'' \in S_{\mathcal{D}_\mathcal{Q}}} \delta_{\mathcal{D}_\mathcal{Q}}(s,s'') \cdot \Pr^{reach}(s'',s') & \text{otherwise.} \end{cases} \tag{2.7}$$

Eq. 2.7 can be solved with any linear equation solver. Finally, the probability of reaching s' is the weighted sum of Eq. 2.7 and the initial state distribution:

$$\Pr^{reach}(s') = \sum_{s \in S_{\mathcal{D}_\mathcal{Q}}} init_{\mathcal{D}_\mathcal{Q}}(s) \cdot \Pr^{reach}(s,s'). \tag{2.8}$$

2.3.3 Expected Time to Reach an Absorbing State

Next, we consider the expected time to reach an absorbing state in a CTMC. Absorbing states have no outgoing transitions, apart from self loops. For a CTMC $\mathcal{Q}$, the set of absorbing states is $S^f = \{s \in S_\mathcal{Q} \mid E(s) = 0 \text{ or } E(s) = \Delta_\mathcal{Q}(s,s)\}$. We solve the expected time problem by model checking the expected cumulative cost until an absorbing state is reached. We use a state-based cost c, which incurs a cost of 1 per time unit spent in a non-absorbing state:

$$c(s) = \begin{cases} 1 & \text{if } s \notin S^f \\ 0 & \text{if } s \in S^f. \end{cases} \tag{2.9}$$

For CTMC $\mathcal{Q}$, the expected cost is computed over the embedded DTMC $\mathcal{D}_{\mathcal{Q}}$ [40]. As $\mathcal{D}_{\mathcal{Q}}$ ignores transition durations, we modify the cost function c to include the expected sojourn time for each state, as defined by the exit rate:

$$c_{\mathcal{D}_{\mathcal{Q}}}(s) = \frac{c(s)}{E(s)}. \tag{2.10}$$

Next, we identify infinite cost states using the CTMC reachability probabilities. Infinite cost states have a non-zero probability of never reaching an absorbing state. From this, we compute the expected cost $\mathbb{E}(s, S^f)$ recursively as the immediate cost plus the weighted cost of the successors:

$$\mathbb{E}(s, S^f) = \begin{cases} 0 & \text{if } s \in S^f \\ \infty & \text{if } \sum_{s' \in S^f} \Pr^{reach}(s, s') \neq 1 \\ c_{\mathcal{D}_{\mathcal{Q}}}(s) + \sum_{s' \in S_{\mathcal{D}_{\mathcal{Q}}}} \delta_{\mathcal{D}_{\mathcal{Q}}}(s, s') \cdot \mathbb{E}(s', S^f) & \text{otherwise.} \end{cases} \tag{2.11}$$

In practice, we use PRISM for efficient CTMC model checking [48]. In Chapter 7, we use the expected time to reach an absorbing state to compute expected task announcement times.

2.4 Phase-Type Distributions (PTDs)

PTDs approximate non-negative continuous distributions using the time taken to reach an absorbing state in a CTMC [49]. We use PTDs to model robot action durations under different interaction scenarios.

Definition 2.8. A *PTD* is a tuple $\mathcal{P} = \langle S, init, \Delta, s^f \rangle$, where S, $init$, and Δ are as in a CTMC. $s^f \in S$ is the single absorbing state such that the probability of reaching s^f is 1. Further, $init(s^f) = 0$.

We denote the set of all PTDs with $\mathbb{P}$. Given a set $\mathbb{P}$ of PTDs, we define the disjoint union of all their state spaces as $S_{\mathbb{P}} = \bigsqcup_{\mathcal{P} \in \mathbb{P}} S_{\mathcal{P}}$. To evaluate PTD $\mathcal{P}$, we use an alternative matrix formulation $\mathcal{P} = \langle \boldsymbol{\alpha}, \boldsymbol{T} \rangle$, where $\boldsymbol{\alpha}$ is a vector containing the initial state probabilities $init_{\mathcal{P}}(s), \forall s \in S_{\mathcal{P}} \setminus \{s^f_{\mathcal{P}}\}$, and $\boldsymbol{T}$ is the *infinitesimal*

generator matrix for non-absorbing states. $\boldsymbol{T}$ captures the rates between non-absorbing states, and the sum of each state's outgoing rates along the diagonal:

$$\boldsymbol{T}(s, s') = \begin{cases} \Delta_{\mathcal{P}}(s, s') & \text{if } s \neq s' \text{ and } s, s' \neq s^f_{\mathcal{P}} \\ \sum_{s'' \neq s} \Delta_{\mathcal{P}}(s, s'') & \text{if } s = s' \text{ and } s \neq s^f_{\mathcal{P}}, \end{cases} \tag{2.12}$$

Using $\boldsymbol{\alpha}$ and $\boldsymbol{T}$, the probability density function (PDF) $f_{\mathcal{P}}$, cumulative distribution function (CDF) $F_{\mathcal{P}}$, and ith moment $\mathbb{E}[\mathcal{P}^i]$ of $\mathcal{P}$ are written as:

$$f_{\mathcal{P}}(x) = \boldsymbol{\alpha} e^{\boldsymbol{T}x} \boldsymbol{t}, \tag{2.13}$$

$$F_{\mathcal{P}}(x) = 1 - \boldsymbol{\alpha} e^{\boldsymbol{T}x} \mathbf{1}, \tag{2.14}$$

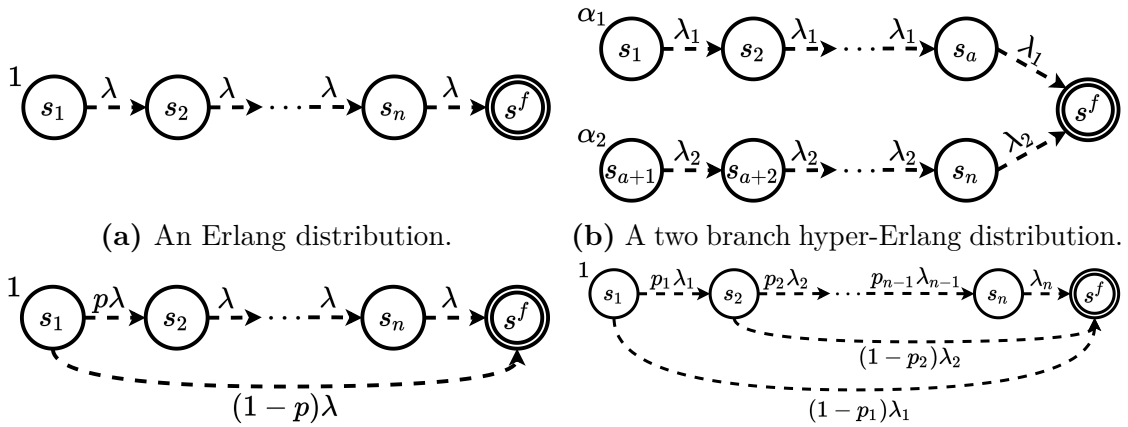
$$\mathbb{E}[\mathcal{P}^i] = i! \boldsymbol{\alpha} (-\boldsymbol{T})^{-i} \mathbf{1}, \tag{2.15}$$

where $\mathbf{1}$ is a vector of ones, and $\boldsymbol{t} = -\boldsymbol{T}\mathbf{1}$ captures the rates to the absorbing state s^f. PTDs are CTMCs, and so the CDF can also be evaluated using transient analysis (see Subsection 2.3.1), and the mean through the expected value computation in Subsection 2.3.3.

PTDs can capture any nonnegative distribution to an arbitrary precision [50], which reduces our assumptions on robot duration distributions. To achieve high precision, we require large numbers of PTD states, which makes evaluation cumbersome. However, in practice sufficient precision can be attained with relatively few states.

2.4.1 Classes of PTDs

By selecting certain classes of PTD, we can improve scalability and leverage efficient fitting algorithms. Fig. 2.5 presents the PTD classes used in this thesis. *Erlang* distributions (see Fig. 2.5a) are a sequence of n identical exponential distributions, and can be evaluated efficiently as a sum of exponentials [49]. We use Erlangs to capture deterministic delays (see Subsection 2.4.3). *Hyper-Erlang* distributions (see Fig. 2.5b) are a weighted mixture of different Erlangs, and can be evaluated using Erlang techniques [51]. We use hyper-Erlangs to represent all simulated duration distributions. *Generalised Erlang* distributions (see Fig. 2.5c) extend Erlangs by adding a direct transition from the initial state to the absorbing state. *Coxian*



(a) An Erlang distribution.

(b) A two branch hyper-Erlang distribution.

(c) A generalised Erlang distribution.

(d) A Coxian distribution.

Figure 2.5: Examples of different PTD structures. Dashed arrows are exponential transitions, and concentric circles are absorbing states.

distributions (see Fig. 2.5d) take this further by adding direct transitions to the absorbing state from all other states [52]. In Chapter 7, we fit generalised Erlangs and Coxians for synthetic experimental environments.

2.4.2 Fitting PTDs from Empirical Data

To construct accurate duration models, we fit PTDs from empirical data representative of execution-time behaviour, which can be achieved with *moment matching* or *maximum likelihood estimation (MLE)*. Moment matching techniques equate the sample and population moments, and then solve the corresponding system of equations to estimate the distribution [53]. Moment matching is efficient, but is insufficient for capturing many distributions. By matching more moments, we can improve the quality of the fit at the cost of increased PTD size, which increases complexity [52]. We use moment matching to construct PTDs for synthetic environments; PTD moment matching is detailed in Appendix C. MLE approaches estimate distributions by maximising the likelihood of observing the sampled data, which often produce better estimates than moment matching [53]. MLE is applied to hyper-Erlangs in [51] using an expectation maximisation algorithm (see Appendix D for details), which we use to fit simulated duration distributions. MLE approaches are widely used, but may fail to find the global optimum, and can be sensitive to

small changes in the sample [53]. Initialising MLE with moment matching can be used to improve distribution estimates [51]. Alternative PTD fitting approaches include [54], which clusters the observations, and then fits a PTD to each cluster.

2.4.3 Approximating Deterministic Delays

Robot actions have a minimum duration due to robot speed and navigation distance etc. For duration distributions, the minimum duration acts as a deterministic delay. PTDs cannot model deterministic delays exactly, and approximating them requires large numbers of states [55]. To mitigate this, [55] proposes modelling the delay as a separate Erlang distribution. For a set of observations $D = \{d_1, ..., d_K\}$, the deterministic delay is the smallest observation $d_{\min} = \min_k d_k$. We fit an Erlang of length n to capture the delay, where each transition has rate $\frac{n}{d_{\min}}$. Increasing n improves modelling accuracy at the cost of increased complexity. We then fit a PTD to the remainder of the distribution by subtracting $d_{\min}$ from each observation, $D' = \{d_k - d_{\min} \mid d_k \in D\}$. The two PTDs are then added by rewiring the absorbing transitions of the first to the initial states of the second, weighted by the initial state probability. We apply a similar approach to add duration distributions in Subsection 6.6.3, and use delay modelling for all simulated duration distributions.

2.5 Markov Automata (MA)

MA extend MDPs to explicitly consider durative events through exponentially timed transitions [56]. We use MA in Chapters 5 and 6 to model asynchronous multi-robot execution, and continuous-time robot execution under time-varying dynamics respectively. We provide an example MA in Fig. 2.6.

Definition 2.9. A *labelled MA* is a tuple $\mathcal{A} = \langle S, \bar{s}, A, \delta, \Delta, AP, Lab \rangle$, where S, $\bar{s}$, and A are as in an MDP, and Δ, AP, and Lab are as in a labelled CTMC. $\delta : S \times A \times S \to [0, 1]$ is an immediate transition function, such that $\delta(s, a, s')$ returns the probability of *instantaneously* transitioning to state s' after executing action a in state s, unlike MDP transitions, which last a single discrete time step.

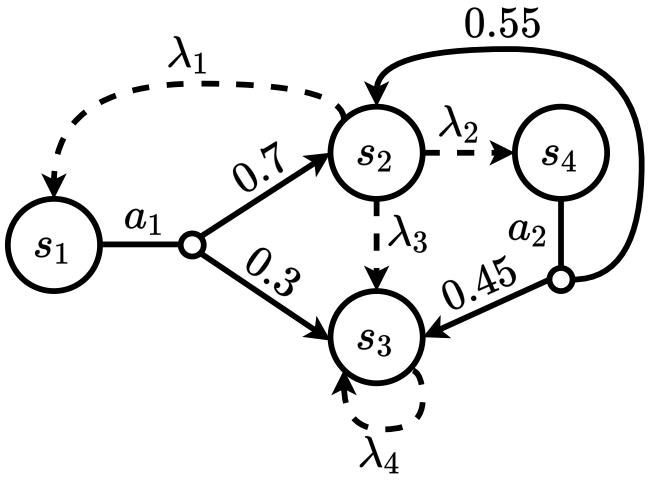


Figure 2.6: An example MA $\mathcal{A}$, where $\bar{s}_{\mathcal{A}} = s_1$, $\delta_{\mathcal{A}}(s_1, a_1, s_2) = 0.7$, $\delta_{\mathcal{A}}(s_1, a_1, s_3) = 0.3$, $\delta_{\mathcal{A}}(s_4, a_2, s_3) = 0.45$, $\delta_{\mathcal{A}}(s_4, a_2, s_4) = 0.55$, $\Delta_{\mathcal{A}}(s_2, s_1) = \lambda_1$, $\Delta_{\mathcal{A}}(s_2, s_3) = \lambda_3$, $\Delta_{\mathcal{A}}(s_2, s_4) = \lambda_2$, and $\Delta_{\mathcal{A}}(s_3, s_3) = \lambda_4$. Solid arrows are immediate transitions, and dashed arrows are exponential transitions.

In an MA, action choices are made at states with immediate transitions, which occur instantaneously. Without loss of generality, we assume no state has both immediate and exponential transitions: immediate transitions fire instantaneously, and so exponential transitions will never fire in states with both transitions. Moreover, we restrict ourselves to MAs that do not exhibit *Zeno behaviour*, where infinitely many immediate transitions can occur in finite time [57]. This prevents us from modelling robots that can perpetually execute instantaneous actions, which is unrealistic. Unlike SSP MDPs, MA do not always have an associated cost function [56–58]. However, in Chapter 6 we define SSPs over time-varying MA (see Section 6.2) by adding a time-varying cost function and a set of goal states.

MA solutions exist for continuous stochastic logic specifications [57], reward-based objectives [59], long-run objectives [20, 58], and multi-objective problems [60]. For untimed properties, the MA is converted into an embedded MDP (similar to CTMCs, see Section 2.3.2), and solved using standard MDP solvers. For timed properties, exponential transitions are often 'digitised' to be of the same duration (similar to uniformisation for CTMCs, see Section 2.3.1), however solution methods are often slow to converge [57].

Algorithm 2.1 Poisson Binomial PMF via Convolution [65].

Input: Random variables $\{X_1, ..., X_n\}$, $X_i \sim \text{Bernoulli}(p_i)$
Output: Vector $\mathbf{PB} \in [0,1]^{n+1}$, where $\mathbf{PB}(j) = \Pr(X_1 + ... + X_n = j)$

$\mathbf{PB} \leftarrow [1 - p_1, p_1]$
for $i \in [\![1, n]\!]$ **do**
    // **Initialise vector of length** $|\mathbf{PB}| + 1$
    $\mathbf{PB'} \leftarrow [0, ..., 0] \in [0,1]^{|\mathbf{PB}|+1}$
    // **Include** X_i **in probability mass function**
    for $j \in [\![0, |\mathbf{PB}|]\!]$ **do**
        $\mathbf{PB'}(j) \leftarrow p_i \cdot \mathbf{PB}(j-1) + (1 - p_i) \cdot \mathbf{PB}(j)$
    end for
    $\mathbf{PB} \leftarrow \mathbf{PB'}$
end for

2.6 Poisson Binomial Distributions

The Poisson binomial distribution models the sum of n independent but differently distributed Bernoulli random variables $\{X_1, ..., X_n\}$, $X_i \sim \text{Bernoulli}(p_i)$. We use the Poisson binomial distribution in Chapter 6 to predict the number of robots present on a topological edge. The probability mass function (PMF) can be evaluated directly at k by considering all instances where the sum equals k [61]:

$$\Pr(X_1 + ... + X_n = k) = \sum_{A \in F^k} \prod_{i \in A} p_i \prod_{j \in [\![1,n]\!] \setminus A} (1 - p_j), \tag{2.16}$$

where F^k contains all length k subsets of $[\![1, n]\!]$. Since $|F^k| = \binom{n}{k}$, Eq. 2.16 quickly becomes intractable [62]. Approximate [63] and recursive [64] approaches have been presented to improve scalability. However, recursive approaches lack stability, and have a high memory overhead [65]. Fourier transform approaches improve upon this for efficient evaluation [62], but are less scalable than the convolution-based approach in [65], which we present in Alg. 2.1.

Alg. 2.1 computes the PMF in $\mathcal{O}(n^2)$ time by sequentially convolving the Bernoulli random variables. For example, we compute $X_1 + X_2 = [(1 - p_1)(1 - p_2), p_1(1 - p_2) + (1 - p_1)p_2, p_1 p_2]$, and then convolve $X_1 + X_2$ with X_3 to obtain $X_1 + X_2 + X_3$. We repeat this sequentially until we obtain $X_1 + ... + X_n$. For large

n, the scalability of Alg. 2.1 can be improved by convolving in the Fourier domain. However, in Chapter 6 n is relatively small, and so Alg. 2.1 is sufficient.

3
Related Work

Contents

3.1 Multi-Robot Planning and Modelling Under Uncertainty 25
3.1.1 Multi-Agent Path Finding (MAPF) 26
3.1.2 Discrete-Time Planning Under Uncertainty 28
3.1.3 Discrete-Time Multi-Robot Planning under Uncertainty 30
3.1.4 Multi-Robot Planning on Single-Robot Models 31
3.1.5 Planning Under Robot Interactions 32
3.1.6 Continuous-Time Planning Under Uncertainty 34
3.1.7 Planning in Non-Stationary Environments 35
3.1.8 Multi-Robot Planning in Partially Observable Environments 36
3.1.9 Multi-Robot Planning Under Resource Constraints . . . 39
3.1.10 Planning versus Reinforcement Learning 40
3.2 Policy Evaluation for Multi-Robot Systems 41
3.2.1 Discrete Event Simulators (DESs) 41
3.2.2 Model Checking Multi-Robot Systems 42
3.2.3 Combining Planning with Model Checking 43
3.3 Multi-Robot Task Allocation (MRTA) 44
3.3.1 Classical MRTA Solutions 44
3.3.2 Auctioning Solutions for MRTA 45
3.3.3 Alternative MRTA Approaches 47
3.3.4 MRTA Under Uncertainty 48
3.3.5 Simultaneous Task Allocation and Planning (STAP) . . 49

In this chapter, we discuss existing approaches for multi-robot coordination. Fig. 3.1 shows the building blocks for coordination and how they relate to each

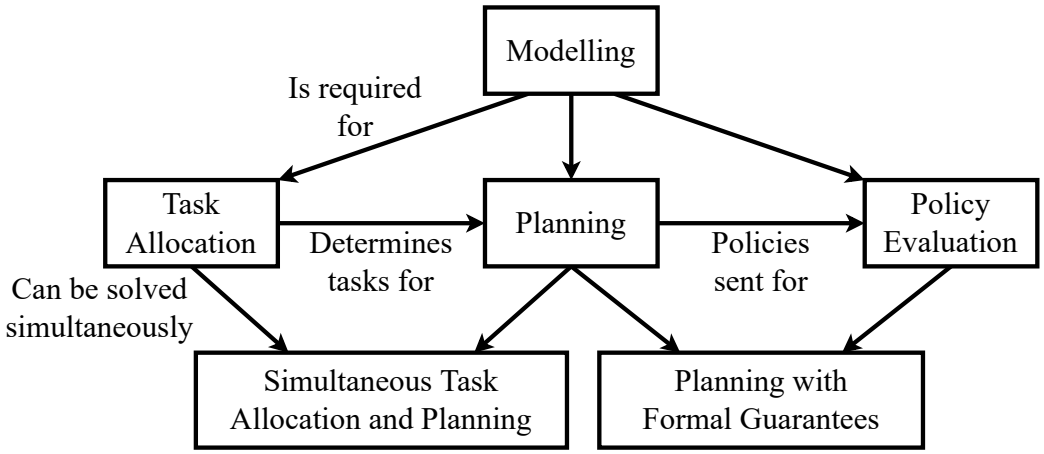


Figure 3.1: The core components required for multi-robot coordination, and the relationships between them.

other. Accurate models of multi-robot behaviour are crucial for planning, task allocation and policy evaluation. We begin by discussing multi-robot planning techniques, and the formal models we plan on. Given a set of policies defining multi-robot behaviour, we then discuss approaches for evaluating these policies on formal multi-robot models. Finally, we discuss MRTA solutions, which dictate the tasks robots plan for, where MRTA methods often use simpler models of robot behaviour for scalability. Each of these building blocks correspond to a research question in Chapter 1. We also discuss techniques for solving these problems jointly, such as simultaneous task allocation and planning, and planning with formal guarantees, which combines planning with policy evaluation.

3.1 Multi-Robot Planning and Modelling Under Uncertainty

Successful planning requires a formal model of the environment which accurately captures robot behaviour. In this section, we discuss approaches for modelling multi-robot systems under uncertainty, and planning techniques which synthesise behaviour over these models. These techniques are key to **Q2 - Modelling** and **Q3 - Planning**.

3.1.1 Multi-Agent Path Finding (MAPF)

We begin by discussing MAPF approaches, which synthesise collision-free paths for robots to reach their goals in deterministic environments [18]. Navigation actions are assumed to be deterministic with fixed durations, and robots execute synchronously, as in MMDPs (see Subsection 2.2.1). Moreover, environments are often discretised into grids. Though deterministic environment models are insufficient for the problems in this thesis, MAPF solvers are widely used for the scalable coordination of multiple robots. In MAPF, robots are assumed to collide if they arrive in the same location simultaneously. This is overly conservative, as motion planners allow robots to manoeuvre around each other in confined spaces [29, 66]. In Chapter 6, we allow robots to be in the same location, and reason over the stochastic effect this has on navigation duration. Under stochastic action durations, the MAPF collision avoidance constraint cannot be guaranteed with probability one, and so MAPF solutions are not suitable for addressing **Q3 - Planning**. However, in Chapter 6 we introduce a baseline which is similar to a MAPF solver, but where two robots can appear in the same location with a small probability.

Cooperative A* is a classical MAPF approach which uses a *reservation table* to store robot route information, which allows robots to avoid those who have planned previously [67]. Each robot plans using A*, where the reservation table constrains the search. Single-robot search can be improved through better heuristics, such as the result of A* in an unpopulated environment, which can be precomputed offline. Moreover, to improve scalability and prevent robots from blocking each other after reaching their goal, cooperative A* can be interleaved with execution online. Cooperative A* scales linearly in the MRS size but is suboptimal, as robots ignore those who are yet to plan. In Chapter 6, we extend the reservation table for actions with continuous stochastic durations, and use the unpopulated environment heuristic to support multi-robot planning under congestion, addressing **Q3 - Planning**.

M* [68] is an optimal MAPF solver, where robots first compute their optimal path ignoring the other robots. These paths are simulated in the joint planning space until a collision is found, at which point we backpropagate through the search

tree, and consider the colliding robots jointly. This improves upon the scalability of planning in the full joint space. In increasing cost tree search (ICTS), search nodes specify a path cost for each robot [69]. Child nodes increase one robot's path cost by one; at each node, ICTS searches for collision-free solutions amongst all paths with the costs in the node. ICTS has been extended for non-unit navigation durations [70], and is optimal. We do not consider adapting ICTS to address **Q3 - Planning**, as under stochastic dynamics robot paths do not have a fixed duration. In conflict-based search (CBS), a high-level search tree maintains robot path constraints, where child nodes increase constraints to prevent collisions [71]. At each node, A* generates robot paths that satisfy the constraints. If the paths are collision-free, CBS terminates at the optimal solution. CBS scalability can be improved through iterative deepening [72], and modified child generation strategies [73]. In Chapter 6, we present a reservation table based approach to **Q3 - Planning**, rather than adapt M* or CBS. We do this to manage the complexity of handling temporal uncertainty, as a reservation table allows us to plan for each robot individually while still capturing the effects of the other robots.

Though MAPF traditionally assumes deterministic environments, uncertainty has been considered. In [74], a belief space is used, giving distributions that appear as 'tails' over the robot's location. M* [68] is then adapted to plan in the belief space of each robot, with coordination occurring between robots likely to collide. In [12], when a robot attempts to navigate, it remains stationary with some probability. To handle action failure, a set of critical dependencies between robots are computed, which force some robots to wait until another robot has reached a certain location. Finally, in [75], M* is adapted to have soft collision constraints, i.e. collisions are allowed to occur, at some cost. Whilst these works consider uncertainty, they do so in ad-hoc manners that consider limited forms of duration uncertainty, which is not suitable for addressing **Q2 - Modelling** or **Q3 - Planning**, where we must accurately capture the effects of temporal uncertainty. The work in [76] improves upon this by bounding discrete navigation durations. CBS is then adapted to find collisions using the intersection of intervals capturing robot presence. However,

this cannot be directly applied to the problems in this thesis, where continuous duration distributions have support across $(0, \infty)$. Later in this section, we discuss rigorous approaches to multi-robot planning under uncertainty.

A few works have considered continuous-time MAPF. In [77], CBS [71] is adapted by replacing A* search for low-level path planning with safe interval path planning [29]. This has been extended to use standard CBS improvements such as conflict prioritisation [78], handle robot kinematic constraints [79], and support online planning and execution [80]. In [81], CBS is reformulated in terms of the satisfiability modulo theories problem to enable the use of efficient SAT solvers. However, [77] and [81] assume that the continuous action durations are deterministic. This is addressed in [82], where continuous, stochastic durations are captured with Gamma distributions. Collisions are now probabilistic, and a modified CBS constrains robots to wait until the collision probability falls below a threshold. The above solutions retain the MAPF collision avoidance constraint. In Chapter 6 we relax this constraint, and reason over the effects of congestion to synthesise efficient robot behaviour, where collision avoidance is handled by a motion planner. We discuss general approaches to continuous-time planning under uncertainty in Subsection 3.1.6.

In Chapter 7, we use auctioning techniques for proactive MRTA. Auctioning can also be applied to cooperative or non-cooperative MAPF, where robots bid on bundles of location-time pairs corresponding to paths [83, 84]. From the bids, an auctioneer assigns the location-time pairs to generate a collision-free solution.

3.1.2 Discrete-Time Planning Under Uncertainty

MAPF approaches fail to consider environmental uncertainty which affects robot execution. For robot planning under uncertainty, MDPs are a common formalism which capture stochastic action outcomes (see Section 2.2). MDPs can be solved exactly using value or policy iteration [85, 86]. However, these methods solve for all states, making them intractable for large models, such as the joint multi-robot models we discuss later.

In Chapter 6, we solve single-robot models with large state spaces, where significant portions of the state space do not contribute towards the optimal solution. Heuristic and sampling-based methods improve upon the limited scalability of value/policy iteration by restricting search to promising areas of the state space. Labelled real-time dynamic programming (LRTDP) samples trials using a heuristic lower bound on the value function [87]. The heuristic guides the search, where states with better heuristic values are explored first. If the heuristic is admissible, i.e. a guaranteed lower bound on the value function, LRTDP converges to the optimal solution in a finite number of trials, which is bounded using the difference between the optimal value function and heuristic. During trials, state values are updated via Bellman backups [10], and LRTDP tracks states whose value has converged to prevent unnecessary backups. We detail LRTDP further in Appendix B, and apply it to single-robot models in Chapter 6. Bounded real-time dynamic programming (BRTDP) extends LRTDP with a heuristic upper bound to explore states with uncertain values [88]. Heuristics are obtained through domain knowledge, or domain-independent techniques such as determinisation [89]. We do not use BRTDP to solve the single-robot models in Chapter 6 as it is difficult to obtain an informative heuristic upper bound for the multi-robot planning problem under congestion. UCT (upper confidence bounds applied to trees) uses upper confidence bounds to balance exploration and exploitation during trials, and a heuristic rollout policy to simulate through unexplored states [90]. UCT can use simulators to sample transitions, which foregoes the need for a model. UCT has been widely used within robotics [91–93], and is effective when no model is available, or when planning is interleaved with execution. In Chapter 6, we solve the congestion-aware planning problem offline, where single-robot models are constructed from duration data obtained from a simulator. As we have access to a model and an informative heuristic, we use LRTDP to efficiently synthesise policies which are robust to the effects of congestion. A key benefit of LRTDP is its convergence condition, which prevents converged portions of the state space from being revisited [94]. UCT has no convergence condition, and may waste samples revisiting unnecessary portions of the state space.

Further, UCT may require many samples to reach the optimal solution. Despite reducing the explored states, heuristic and sampling-based algorithms are slow to converge on large models, but often provide anytime behaviour such that valid solutions are synthesised quickly, and improved with time.

In discrete-time MDPs, robot dynamics are simplified to assume all actions have an identical fixed duration, similar to MAPF approaches, which fails to capture execution-time robot behaviour. Timed MDPs relax this assumption using discrete duration distributions [95], and are a subset of semi-MDPs, which allow for continuous durations. We discuss semi-MDPs alongside other approaches for continuous-time planning in Subsection 3.1.6.

3.1.3 Discrete-Time Multi-Robot Planning under Uncertainty

In this thesis, we model and coordinate *multiple* mobile robots under uncertainty. MRSs can be modelled using MMDPs (see Subsection 2.2.1), which can be solved with MDP solution methods. However, the solution methods in Subsection 3.1.2 converge slowly on large models, and MMDPs scale exponentially in the team size, making optimal planning intractable. This can be improved by simplifying the model. For example, transition-independent MMDPs [96] and constrained MMDPs [97] assume robots are independent, and coupled through rewards and shared resources respectively. Team MMDPs [98] also treat robots independently, modelling them sequentially to solve simultaneous task allocation and planning problems (see Subsection 3.3.5). Independence assumptions decouple the robots for more efficient solution methods. For example, if the robots are entirely independent, each robot can be modelled as an MDP and solved separately. However, independence assumptions often do not hold in practice, where execution-time interactions such as congestion affect the outcome and duration of robot actions, which makes them unsuitable for reasoning over temporal uncertainty and addressing **Q3 - Planning**. This can be mitigated by augmenting single-robot planning models with knowledge of the other robots, as we discuss in Subsection 3.1.4.

As introduced in Subsection 2.2.1, MMDPs enforce synchronisation, where all actions have the same fixed duration, similar to MDPs. Consider a cost minimisation problem, where cost corresponds to the duration of robot tasks. Under synchronous execution, all robots have to wait for each other to finish acting before a subsequent action can be performed [22]. This increases duration, and thus the cost, as robots wait unnecessarily, when they could be continuing execution. Therefore, synchronous execution leads to suboptimal execution-time behaviour. Moreover, robot action durations are inherently continuous and uncertain, where robot interactions contribute towards this uncertainty. To address **Q3 - Planning** and synthesise efficient multi-robot behaviour, we must capture asynchronous multi-robot behaviour. For this, we must model uncertainty over action duration, which requires a continuous-time Markov model. We discuss approaches for continuous-time planning under uncertainty in Subsection 3.1.6, and asynchronous multi-robot planning under partial observability in Subsection 3.1.8.

Markov games generalise MMDPs to allow competitive agents with opposing reward functions [99, 100]. For fully cooperative agents sharing the same reward, Markov games are equivalent to MMDPs [101]. Markov games are joint models, and scale similarly to MMDPs, while making the same assumptions over robot behaviour. Markov games can be solved using MMDP solution methods adapted to competitive settings [99].

3.1.4 Multi-Robot Planning on Single-Robot Models

For the problems addressed in this thesis, execution-time interactions such as congestion affect robot navigation duration, introducing dependencies between the robots. To avoid the scalability of planning on joint models while still considering robot dependencies, it is possible to plan using single-robot models that are extended to include some knowledge of the other robots. In [102], robots learn heuristic models of each other from simulation data, and use them to obtain single-robot policies with UCT. Robots also plan on single-robot models in [21], using a procedure where robots consider their team mates more with every iteration. During planning, a

cost function captures the probability of robot collisions under stochastic navigation durations, as well as action synergies, where one robot's action helps another robot. However, action execution is assumed to be deterministic, with the output being a sequence of actions, which is less robust to the probabilistic nature of the environment. In [103, 104] single-robot models aggregate the responses of the other robots assuming they are only aware of themselves. The aggregate response is represented as a distribution which predicts whether any robot is present at a given location. This is computed by combining individual distributions over each robot's location. We take a similar approach in Chapter 6, where we approximate multi-robot behaviour by creating single-robot models which capture the probabilistic effects of congestion. We compute distributions over congestion, which combine the probabilities that each robot is present at a topological edge to produce a distribution over the number of robots present.

In Chapter 6, we generate policies for each robot *sequentially*, with each robot only considering those that planned before it. Such an approximation is common in multi-robot planning [98, 105], such as in cooperative A* [67], as discussed in Subsection 3.1.1. The quality of plans produced under a sequential planning assumption is sensitive to the ordering of robots [67], and determining an optimal ordering is NP-hard [106]. However, effective priority orderings have been generated using heuristics [107, 108], optimisation methods [109], and path prospects [110], where robots with fewer path options have higher priority.

3.1.5 Planning Under Robot Interactions

Assuming robots are completely independent simplifies planning but fails to capture execution-time robot behaviour, as discussed in Subsection 3.1.3. However, for many multi-robot problems, robots can act independently for the majority of execution, as interactions are sparse. For example, two robots conducting a handover can ignore each other until they are close. Further, for the congestion-aware planning problem in Chapter 6, interactions only occur between robots at nearby map resources. Solutions for interaction-driven Markov games (IDMGs) [101] and decentralised

sparse interaction MDPs (Dec-SIMDPs) [111, 112] exploit this to improve upon the complexity of solving MMDPs. IDMGs and Dec-SIMDPs are equivalent, and use an independent MDP per robot, and a set of interaction MMDPs, which define joint MRS behaviour in interaction areas, such as near a doorway. Though interaction MMDPs are joint models, they are significantly smaller than the full MMDP as they are defined over only a small fraction of the full MMDP state space. Solutions exploit this structure by solving the single-robot MDPs and interaction MMDPs separately using standard solvers such as value iteration [101]. The model can be relaxed further by assuming transition independence [96], however this assumes interactions do not affect robot action outcomes. IDMGs and Dec-SIMDPs assume that interactions are localised to a small, fixed part of the environment. This does not hold for the congestion-aware planning problem we solve in Chapter 6, where interactions such as congestion are sparse but may occur anywhere. This reduces IDMGs and Dec-SIMDPs to the full MMDP, which are intractable to plan on with optimal solvers. The locality assumption is avoided in [21], where the plan cost captures the effects of collaborative interactions, wherever they occur. In Chapter 6, we augment single-robot models with distributions over congestion. If there is no congestion, robots act independently; if there is, robots reason over congestion for robust policy synthesis.

In Chapter 6, we focus on interactions due to congestion, which is often modelled in terms of the incurred time cost. In [113], different duration distributions are applied at different intervals to capture congestion variation for vehicle route planning. The effects of human congestion on mobile robots is modelled in [114] using Fourier analysis or Gaussian mixture models. The probability of congestion is then used to penalise robot travel time. In [115], electric vehicles share probabilistic information about their arrival to charging stations with a centralised entity, which then computes discrete distributions over charging station congestion. These distributions are shared with the vehicles, which allow them to improve their policies. We take a similar approach in Chapter 6, where a centralised probabilistic

reservation table stores CTMCs describing robot policy execution, from which we compute congestion distributions to support planning on single-robot models.

3.1.6 Continuous-Time Planning Under Uncertainty

To solve **Q2 - Modelling** and **Q3 - Planning**, we require continuous-time models of multi-robot execution. Robot actions exhibit continuous durations, where there is zero probability of observing the same duration twice, due to variance in localisation error or weather conditions etc. Therefore, continuous-time planning models must capture stochasticity in robot action durations. We discuss phenomena which affect action durations further in Chapter 4. Semi-MDPs use arbitrary duration distributions for each state/action pair [116], and have been used to model underwater gliders [91]. Temporally extended actions are represented as options, where MDP solutions such as value iteration can be adapted to synthesise option policies. However, semi-MDPs allow only one durative action to be executed at a time, which is similar to an MA if only one exponential transition could be enabled in a state (see Section 2.5). This limits the use of semi-MDPs for modelling MRSs, where robots execute actions concurrently and asynchronously, and so we do not use them in this thesis. In contrast, continuous-time MDPs (CTMDPs) enable concurrency, but restrict durations to exponential delays [117]. CTMDPs have been used for probabilistic control [118] and MRSs [119]. To solve CTMDPs for time-abstract objectives, such as expected untimed reward, MDP solvers are applied to the embedded discrete-time MDP computed from the CTMDP branching probabilities, as described for CTMCs in Subsection 2.3.2. For timed objectives, MDP solvers are instead applied to the uniformised MDP, computed using the techniques in Subsection 2.3.1 [52, 120, 121]. As with MMDPs, CTMDPs scale exponentially in the number of robots. To mitigate this, [119] constructs single-robot CTMDPs, assuming transition independence. Each action in a CTMDP is associated with a single exponential distribution, which limits the accuracy with which we can capture robot action durations. Therefore, in Chapters 5 and 6 we use MAs, which explicitly separate robot action choice and the duration of

robot actions. This allows robot action durations to be represented as PTDs (see Section 2.4), i.e. a sequence of exponentials, which can capture any nonnegative distribution to an arbitrary level of precision [49].

Generalised stochastic Petri nets (GSPNs) are a more compact formalism for homogeneous MRSs, where robots are anonymous, and durations are restricted to exponentials [122]. GSPNs remain exponential in the team size, but robot anonymity provides a practical reduction. GSPNs have been used for modelling, analysis and planning for MRSs under uncertain action durations [19, 20, 123]. Further, GSPN semantics can be described with an MA, which we introduced in Section 2.5 [124]. Due to robot anonymity, GSPNs cannot model the planning problem we introduce in Chapter 6, where each robot has its own goal. Generalised semi-MDPs (GSMDPs) also handle concurrency and stochastic durations, and have been applied to MRSs [22, 52], but are complex to define, and hard to solve, as GSMDPs allow for arbitrary duration distributions. In fact, a standard solution method involves converting all duration distributions into PTDs [49], which produces a model similar to the multi-robot MA we propose in Chapter 5 [52]. MA are simpler to define, and can be solved directly [57], as all durations are exponentials/PTDs by definition. Therefore, we use MA in Chapters 5 and 6 to capture asynchronous multi-robot behaviour in continuous time. This presents a unified approach to **Q2 - Modelling**, as MAs, PTDs, and CTMCs all use exponentials to capture durative events. With this, we can insert PTDs directly into MAs to capture robot action durations (see Subsection 5.2.2), and construct CTMCs which capture the continuous-time behaviour induced by a robot's policy (see Subsection 6.6.3).

3.1.7 Planning in Non-Stationary Environments

Robot environments are commonly assumed to be static. However, for MRSs, the presence of other robots introduces non-stationarity. Consider two warehouse robots, where aisles in the warehouse are only wide enough for a single robot. If the robots do not know each other's location, then from the perspective of a single robot, its ability to traverse certain aisles changes with time, i.e. the duration and

outcome of actions are not stationary. In Chapter 6, we capture the non-stationary temporal dynamics of an MRS using distributions over congestion. There exist Markov models which capture non-stationary transition dynamics. Non-stationary MDPs model time-varying transitions and rewards, but assume time is discrete [125]. Time-dependent MDPs can capture continuous-time dynamics and durative actions, however in practice rewards and durations are assumed to be piecewise linear and discrete respectively [126]. This produces a value function which is a piecewise linear function of time at each state, which allows Bellman backups to be computed exactly, and policies to be synthesised with value iteration. However, time-dependent MDP solutions scale poorly [127]. Time-varying MDPs index the transition and reward functions with time [128], and are solved by alternating between traditional Bellman backups, and solving systems of linear equations to estimate the time each state is reached under the current policy. However, this relies on deterministic models of action duration, though recent work extends this for duration variance [129]. The above simplifications are not applicable to our work, as we aim to explicitly model and reason about the uncertainty over action duration. Recent work [91] has begun to address these issues. There, an approximate solution to time-varying semi-MDPs is presented, where non-stationary transition dynamics are learned online with a Gaussian process, and planning is carried out by extending Monte Carlo tree search to consider non-stationarity and stochastic transition durations. However, this assumes only one action is being executed at a time, which does not hold in MRSs, where robots execute actions concurrently and asynchronously. In Chapter 6 we introduce time-varying MAs, which extend time-varying MDPs [128] to capture continuous action duration distributions.

3.1.8 Multi-Robot Planning in Partially Observable Environments

In some MRSs, robots only *partially observe* the environment, as limited communication and sensing capabilities prevent robots from knowing each other's states or the global state features. To solve partially observable problems, we can plan

on partially observable MDPs (POMDPs), where robots make local observations which update their belief over the current state [11]. POMDPs have been used for robotic manipulation [92], navigation [130], and task allocation [131]. In Chapter 6, we solve **Q3 - Planning** using single-robot models which capture the effects of congestion. We assume robots can fully observe their local state during execution, and do not communicate with each other. An alternative approach is to plan on single-robot POMDPs, where each robot maintains a belief over the states of the other robots using observations of their location. POMDPs can be represented as a fully observable MDP over the continuous belief space, but this is challenging to solve [11]. Point-based methods are a more practical planning approach, where computation is restricted to a finite subset of reachable beliefs, and the value function is approximated with a set of linear functions [132]. A bottleneck in POMDP planning is computing exact Bayesian belief updates. This is mitigated in [133], where UCT [90] is extended for POMDPs. There, the current belief is represented as a particle filter, and belief updates are computed through Monte Carlo sampling. We do not use POMDPs in this thesis due to the complexity of maintaining beliefs, and because action durations are assumed to be deterministic.

Decentralised POMDPs (Dec-POMDPs) extend POMDPs to multi-robot settings [134], and have been used for warehouse robotics [135], cooperative package delivery [136], and teams of unmanned aerial vehicles [137]. Dec-POMDPs can be solved centrally to synthesise local policies for decentralised execution, which map from local action-observation histories to actions. If the combined local observations uniquely identify the joint state, Dec-POMDPs are reduced to Dec-MDPs, which exhibit lower complexity [134]. However, synthesising optimal solutions on both Dec-POMDPs and Dec-MDPs is more complex than MMDP planning [134], and so is intractable for most realistic problems, including the congestion-aware planning problem in Chapter 6. To improve scalability, [138, 139] decouple Dec-POMDPs into local POMDPs for each robot. Rather than assuming robot independence, each robot computes a distribution which captures how external state factors influence its local state. These external state factors include the states of the other

robots. This is then used to marginalise out the external state factors to construct a single-robot POMDP. Influence-based abstraction improves model complexity, but computing influence distributions is intractable in general [139], making it unsuitable for the problems in this thesis.

To solve **Q2 - Modelling**, we require models which capture asynchronous multi-robot execution. Macro actions [140] are temporally extended actions captured with the options framework [116], which improve scalability by leveraging existing behaviour such as navigation, and reasoning at the macro action level, rather than each time step. MacDec-POMDPs extend Dec-POMDPs to reason over macro actions, where macro actions are executed asynchronously, and synchronisation occurs at the level of the controllers that form the macro actions [140]. Dec-POMDP solutions can be adapted to MacDec-POMDPs to synthesise policies over macro actions. In [141], the space of macro-action policies is searched exhaustively, where efficient simulators improve the scalability of policy evaluation [135]. This approach scales poorly, and is addressed in [142], where a heuristic search method optimises compact finite state controllers for each robot. Macro actions execute a series of primitive low-level actions, such as moving one grid cell forward. Every MacDec-POMDP has an underlying Dec-POMDP which reasons over these low-level actions. MacDec-POMDPs assume that the underlying Dec-POMDP and the policies for each macro action are known [143]. Decentralised partially observable semi-MDPs (Dec-POSMDPs) relax this assumption by reasoning over macro action completion times, where solution methods are similar to those for MacDec-POMDPs [136]. MacDec-POMDPs and Dec-POSMDPs can capture the congestion-aware planning problem in Chapter 6 by using a macro action per topological edge, and allowing robots to observe each other's locations to determine the current congestion level. However, MacDec-POMDP and Dec-POSMDP solutions have not been shown to scale beyond teams of around four robots [136, 140], and so we do not use them in this thesis. In Chapter 6 we plan on single-robot models which summarise the effects of the other robots using congestion distributions. This approach scales to teams of 15 robots (see Subsection 6.8.2).

3.1.9 Multi-Robot Planning Under Resource Constraints

Recall from Subsection 3.1.3 that constrained MMDPs model independent robots coupled by resource constraints, which can capture robots experiencing congestion, or task allocation for example [97]. Constrained MMDPs support different types of constraint, such as soft constraints, which must be satisfied in expectation, and hard constraints, which must never be violated. A common constrained MMDP solution approach is to solve a dual linear program (LP) which optimises stochastic policies for each robot [97]. Alternatively, column generation uses the solution to a master LP to modify single-robot reward functions to consider the constraint [144]. Single-robot policies are then fed back into the master LP. An online approach is found in [145], where robots plan independently, and during execution, an arbiter detects imminent constraint violations and modifies the joint action to prevent it. Further, each robot models the effects of the arbiter, and reasons over it during planning. Resource bounds are often fixed, but in [15], they evolve stochastically according to a DTMC, such that there is a different constraint at each discrete time step. If the resource constraint is hard, a mixed integer LP is used to preallocate resources to robots at each time step. For soft resource constraints, a standard constrained MMDP LP formulation is extended to handle the evolving constraint. In both solutions, each robot considers the product of its local state with the current constraint state. Similarly to [15], in Chapter 7 we use CTMCs to represent stochastic task announcement. Most constrained MMDP solutions optimise for soft constraints. In [146], soft constraint solvers are used to solve for hard constraints through frequent online replanning. Finally, chance constraints are considered in [147] by iteratively relaxing the resource bound to obtain a chance constrained solution for the original bound. Constrained MMDPs can partially represent the congestion-aware planning problem in Chapter 6, where constraints on topological edge usage capture the effects of congestion. However, constrained MMDPs are discrete-time models, and solution methods require robots to execute synchronously, which is unsuitable for synthesising efficient and asynchronous multi-robot behaviour in continuous time.

3.1.10 Planning versus Reinforcement Learning

In Chapter 6, we use planning techniques to synthesise multi-robot behaviour under congestion. An alternative approach to policy synthesis is reinforcement learning (RL) [148]. Planners synthesise behaviour using a model of the system, whereas RL approaches learn behaviour using data sampled from the environment [37, 148]. RL can be model-based, where a planner is applied to a learned model, or model-free, where a policy is learned directly from data. There is significant overlap between planning and RL. For example, the coordination approach in this thesis is similar to offline model-based RL [149], where model learning and planning occurs offline. In Chapter 4, we collect duration data describing the effects of robot interactions; in Chapter 5 we construct multi-robot models from this data; and in Chapter 6, we plan on single-robot versions of these models to synthesise multi-robot behaviour. However, our planning models are constructed manually, not learned, which allows domain knowledge to be encoded into the model, such as regularity among topological edges (see Section 4.4). This helps avoid model exploitation [149], where a planner exploits the parts of a learned model which are not sufficiently captured by the data, which causes poor execution-time performance.

Deep RL methods achieve state of the art performance on many complex problems [150] such as robotic manipulation [151], legged locomotion [152], and StarCraft [153], but often require millions of samples to converge. This is cumbersome for robotics problems where interacting with the environment is expensive. In Chapter 5 we present CAMAS, which can be used as a model-based surrogate to reduce the cost of sampling from the environment while still capturing multi-robot behaviour. Deep RL methods are also hard to scale to multi-robot problems, as the state and action space increases exponentially in the number of robots [153], similar to joint multi-robot planning models (see Subsection 3.1.3). This can be addressed by decoupling the robots while still considering the value of joint actions [153]. The poor scalability of deep RL often makes it unsuitable for problems where new tasks appear online, such as the MRTA problem in Chapter 7, where replanning is significantly

cheaper than retraining an RL agent. However, goal-conditioned RL techniques can improve upon this by conditioning a robot's policy on its current goal [154].

3.2 Policy Evaluation for Multi-Robot Systems

In this section, we discuss simulation and model checking techniques, which evaluate task-level metrics of multi-robot performance given a model of the system and a set of policies defining robot behaviour. In this thesis, we apply both of these techniques to address **Q4 - Policy Evaluation**.

3.2.1 Discrete Event Simulators (DESs)

Simulators evaluate multi-robot behaviour by executing the robot policies in an abstracted model of the environment. Physics-based simulators such as Gazebo [155], Webots [156], CoppeliaSim [157] and RaiSim [158] accurately reproduce system dynamics for realistic simulations. However, the computational overhead incurred when simulating complex MRSs forces the simulator to run close to real time, which is cumbersome for multi-robot policy evaluation [159]. Therefore, we do not use physics-based simulators to solve **Q4 - Policy Evaluation**. However, in Chapter 5 we validate our multi-robot models and simulator against Gazebo [155]. DESs such as the CAMAS simulator we propose in Chapter 5 mitigate the complexity of physics-based simulators by abstracting away low-level robot dynamics [160], allowing simulations to run magnitudes faster than real time. DESs typically use an event queue, which facilitates asynchronous robot execution by ordering events based on occurrence time [161]. DESs which sample stochastic robot action durations have been limited to Gaussians or exponentials [6, 162, 163]. Throughout this thesis, we use PTDs to model stochastic action durations, which approximate nonnegative distributions to an arbitrary precision [51] (see Section 2.4). In a DES, robot behaviour may be fixed [159], specified as finite state machines [6], presented as generic software libraries [160], or provided as a set of policies over the joint MRS state [163], similar to CAMAS.

3.2.2 Model Checking Multi-Robot Systems

Model checking techniques evaluate the behaviour induced by robot policies by systematically checking if a property is satisfied in a formal robot model [164]. Properties are often specified with temporal logics, such as linear temporal logic (LTL) or continuous stochastic logic (CSL). LTL model checking techniques compute a product of the model with an automaton that captures the formula. However, the product significantly increases the state space, and is unsuitable for multi-robot problems. CSL describes continuous-time properties, such as time-bounded reachability, which can be model checked on CTMCs using transient analysis, as described in Section 2.3. In Chapter 6, we apply these techniques to evaluate timed properties of robot behaviour after policy synthesis. Alternating-time temporal logic (ATL) extends computation tree logic (CTL) for MRSs by quantifying properties over sets of agents [165], such as 'can these robots complete their objective irrespective of the other robots'? ATL model checking techniques recurse over the sub-formulae, similar to CTL model checking. Moreover, ATL has been extended for resource constrained and probabilistic MRSs [166–168]. ATL model checking requires a joint model of the MRS, however no techniques exist for model checking ATL formulae on MAs. Therefore, we do not apply ATL model checking to the multi-robot MA in Chapter 5 or the single-robot models in Chapter 6.

Bounded model checking techniques unroll a model for a fixed number of steps sufficient for verification, and utilise efficient SAT solvers [169]. This improves scalability, and has been used for MRSs in [170]. Scalability can also be improved through abstraction. In [171] each agent's states, actions and transitions are grouped into equivalence classes to decrease the model size. Equivalence classes are defined such that satisfaction on the abstracted model corresponds to satisfaction on the original model. By parameterising an MRS, we allow for model checking over a range of conditions. The work in [172] analyses whether a property holds irrespective of obstacle locations. For unknown team sizes, [173] finds a cut off point where property satisfaction is unchanged for any larger number of robots. Therefore,

model checking can be limited to teams no larger than the cut off, which reduces the state space. This has been extended to evaluate fault tolerance [174].

Statistical model checking (SMC) techniques evaluate properties by sampling through a model given a set of robot policies, which avoids enumerating the state space [175], and bridges the gap between simulation and model checking techniques. In [176], SMC is used to evaluate quantitative properties of an MRS. In Chapter 5, we sample through an MA to simulate multi-robot behaviour, similar to SMC techniques. SMC is applied to MA in [177], where sample termination conditions are given for bounded and unbounded properties. A drawback of SMC is a possible failure to explore states reached with low probability, which can render SMC unsuitable for safety critical systems [177].

Numerous multi-robot model checking tools are available, including MCMAS [178], MABLE [179], and PRISM-Games [180]. In [181], simulation and model checking are combined to verify human-robot teams, which increases confidence in our predictions of multi-robot behaviour.

3.2.3 Combining Planning with Model Checking

Model checking and planning can be combined to synthesise guaranteed multi-robot behaviour. For temporal logic specifications, we can plan over a joint product automaton, however this quickly becomes intractable. The work in [98] concatenates single-robot product automata through switch transitions in a team MMDP to reduce the state space. As previously discussed, a common MRS planning approach is to augment single-robot models with an abstracted knowledge of the team. This is extended in [182], where single-robot product automata capture the other robots as a single adversarial meta-agent. Each robot then identifies behaviours where other robots prevent it from completing its task. These behaviours are then exchanged between robots, who update their planning models to guarantee they do not enact them. Single-robot product automata are also used in [183], but if a consistent set of plans are not found planning resorts back to the joint model. In [184], robots

are incrementally added to the product automaton until the full problem is solved, or a fixed computational budget is exceeded.

Product automata are avoided in [185], where robots synthesise policies that achieve a single automata transition. Similarly, in [186, 187], robots generate control laws to satisfy an automaton transition or atomic proposition using control barrier functions. Multi-robot temporal logic planning can also be formulated as an LP. In [188, 189], temporal logic formulae are converted into mixed integer linear programming constraints, which can be solved with standard techniques.

To avoid fully constructing the product automaton [190–192] explore it incrementally through sampling. RRT* [28] is modified to sample through the product space and synthesise satisfying motion plans. This sampling-based approach scales to problems previously unsolvable [191], but assumes deterministic environments. However, RRT* has recently been extended to MDPs [193], and could be combined with the temporal logic techniques in [190–192] to synthesise guaranteed behaviour in stochastic environments. If applied to the models presented in this thesis, combined planning and model checking techniques could address **Q3 - Planning** and **Q4 - Policy Evaluation** simultaneously. However, we decouple these problems due to the increased complexity of considering them jointly, such that we can efficiently synthesise multi-robot behaviour while capturing temporal uncertainty.

3.3 Multi-Robot Task Allocation (MRTA)

In this section, we discuss foundational MRTA techniques such as auctioning, which we use extensively in Chapter 7 to address **Q5 - Task Allocation**. Further, we discuss progress on MRTA in stochastic environments, and simultaneous task allocation and planning.

3.3.1 Classical MRTA Solutions

MRTA problems where each robot executes one task can be formulated as an LP, and solved using standard techniques [194]. Alternatively, the Hungarian algorithm models robots and tasks in a bipartite graph, and iteratively finds a

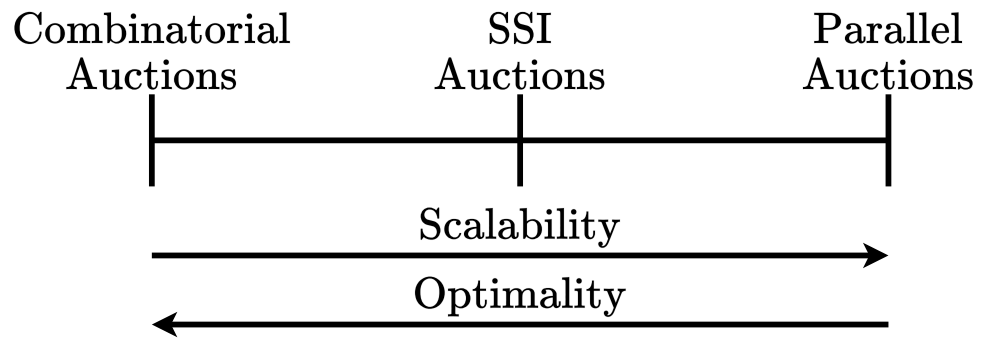


Figure 3.2: The spectrum of auctioning solutions.

minimal cost matching [195]. These methods are optimal, but scale poorly, as they are fully centralised. Therefore, these techniques are not suitable for solving **Q5 - Task Allocation**, where stochasticity further increases complexity. The Hungarian algorithm has been adapted to decentralised MRSs, which improves scalability in practice [196]. A simple greedy heuristic for MRTA is to iteratively select the robot and task with minimum cost [194], which scales to large problems, but often performs poorly.

3.3.2 Auctioning Solutions for MRTA

Auctions are a distributed MRTA method where robots bid the cost they can offer for each task to an auctioneer who determines the allocation [197]. Computation is distributed across the robots, improving efficiency over fully centralised methods. Moreover, auctioning can be adapted to different objectives without adjusting the auction structure, as bid computation and winner determination are modular. We utilise this in Chapter 7 to compute bids for tasks with stochastic announcements, both in space and time, addressing **Q5 - Task Allocation**. However, auctions are often run by a centralised auctioneer, which is a single point of failure. Decentralised auctions improve robustness by allowing robots to reach a consensus over the allocation without a central auctioneer [198]. In Chapter 7 we use a centralised auctioneer, which has lower communication requirements than decentralised approaches, as robots only communicate with the auctioneer.

Auctioning techniques trade between optimality and scalability, as seen in Fig. 3.2. In combinatorial auctions, robots bid on task bundles representing possible schedules [199], and the auctioneer assigns the bundles with minimum combined cost. Combinatorial auctions are optimal but intractable, as the number of bundles increases exponentially with the tasks. In parallel auctions [200], robots compute a bid for each task, which is assigned to the robot with the lowest bid. This is highly scalable but synthesises poor solutions, as spatial dependencies between tasks are ignored, i.e. we ignore that two nearby tasks may be efficiently completed by the same robot. Sequential single-item (SSI) auctions improve on this by running a multi-round auction, where one task is allocated per round [200]. In each round, robots submit bids for each task, which represent the increase in schedule cost if allocated that task. The auctioneer allocates the task with minimum bid, and the winning robot recomputes its bids for the remaining tasks. Recomputing bids allows the robot to capture cost changes after being allocated a task, which may represent spatial dependencies. SSI is scalable, as each round is as complex as a parallel auction, where there is a round per task. However, SSI is sub-optimal, but is at most a factor of two from the optimal for sum of cost objectives [200]. In Chapter 7 we use SSI to proactively allocate tasks, as it effectively trades between scalability and solution quality. The SSI algorithm is described in detail in Appendix E.

SSI has been widely extended, such as for online deployment, where new tasks are inserted into a robot's existing schedule [201]. To improve allocations, [202] evaluates the future impact of assigning a task by increasing the SSI lookahead, at the cost of additional complexity. Further, in [203] robots bid on fixed size bundles, bringing SSI closer to combinatorial auctions. SSI with regret clearing assigns the task with maximal regret, i.e. the task with the largest gap between the two smallest bids, which improves performance [204]. These techniques could be applied to the SSI framework we present in Chapter 7 to improve performance, however this is left to future work. In [205], robots maintain simple temporal networks to ensure tasks are completed within a time window. This has been extended for precedence constraints, where tasks with longer precedence chains have higher priority [206].

SSI has been applied to human-aware navigation in [114], where robot bids are penalised for navigating through congested areas.

Though we use a centralised MRTA method in Chapter 7, the techniques we present can be adapted to use a decentralised MRTA solution in a straightforward way. Typically, decentralised auctions are iterative processes, where robots compute bids, communicate them with each other, and update their bids until a consensus is reached [198, 207–209]. In [131], robots compute local bids at each decision step and send them to the other robots, where each robot finds an allocation separately using the Hungarian algorithm [195]. An alternative approach is to use robots as auctioneers for different tasks. In [210] and [211], robots reauction tasks prior to execution and upon completion respectively to validate their schedules. In [212], robots execute tasks in a self-interested way, and only call auctions if collaboration may increase reward. Tasks may require multiple robots, and in [213], robots are grouped by the auctioneer based on a task's required capabilities. In [214], complex tasks are represented as trees, and robots bid on subtrees representing subtasks. Finally, in [215], robots use the firefly algorithm to determine which task to bid on, and the auctioneer selects a task coalition using genetic algorithms and artificial bee colony optimisation.

3.3.3 Alternative MRTA Approaches

We use auctions to address **Q5 - Task Allocation** because of their performance and flexibility. However, alternative MRTA approaches have been presented. Greedy heuristics such as earliest deadline first and nearest task first [216] are highly scalable, but synthesise poor solutions. Auctions make numerous assumptions, such as a robot's ability to accurately evaluate tasks. This is mitigated in [217], where 'bondsmen' announce tasks with a time-increasing reward, and self-interested robots compete to finish tasks and claim the reward. In [218], tasks are clustered to balance tasks across the MRS, where robots evaluate clusters by solving a travelling salesperson problem.

Alternative decentralised approaches include [219], where robots form distinct groups based on communication constraints. In each group, a single robot computes an allocation for the tasks the group is aware of. In [220], robots iteratively broadcast their best task along with a slack value which represents the cost gap between its two best tasks. If multiple robots broadcast the same task, the robot with lowest slack changes task.

For multi-robot tasks, in [221] robots maintain and communicate local sets of coalitions, where robots iteratively adjust their chosen coalition based on information received from the other robots. Alternatively, in [222], coalitions are merged and split based on task resource requirements, where coalitions are compared in terms of Pareto dominance over robot cost vectors. Coalition formation can also be solved using genetic algorithms, where each individual represents a possible solution [223].

3.3.4 MRTA Under Uncertainty

To synthesise robust allocations, MRTA methods must reason over uncertain robot dynamics and task announcement. To solve **Q5 - Task Allocation**, MRTA solutions must be robust to sources of temporal uncertainty which affect robot action durations and task announcement times. Tasks can be reallocated reactively to stochastic dynamics online, such as in [224], where tasks are reallocated upon human behaviour changes. This allows for improved allocations using updated information. However, such methods ignore stochasticity during allocation, generating inefficient allocations, as expectations of task execution diverge from what we observe during execution. In Chapter 7, we combine online reallocation with an MRTA algorithm that explicitly considers stochastic task announcements. The work in [225] optimises the expected number of completed tasks using the probability of robot task failure. MDPs have been used for MRTA in [98, 185] to allocate temporal logic tasks. In [131], each task is represented with a POMDP, which each robot solves to obtain bidding functions for a decentralised auction. To reduce the effects of duration uncertainty on task completion times, [226] and [227] assign redundant robots to tasks, which increases the chance of a robot arriving quickly. However, in many MRTA problems,

tasks outnumber robots, and so there is no redundancy to exploit. For chance constrained MRTA, [228] splits the constraint across the robots, where each robot's risk threshold is computed assuming robot costs such as task durations are Gaussian.

Stochastic task announcement is fundamental to **Q5 - Task Allocation**, and can be handled by allocating tasks as they are announced [225, 229]. However, this poorly utilises the MRS, as robots are idle until task announcement. In [230, 231], tasks are announced uniformly across the environment according to a Poisson distribution. However, tasks are only allocated once they are announced. To increase MRS utilisation, tasks should be allocated proactively to reduce the duration between announcement and the task being service. This is at the core of our solution to **Q5 - Task Allocation** in Chapter 7. In [103], task announcement distributions allow robots to anticipate new tasks during planning. Robots select tasks independently by predicting the tasks selected by other robots using approximations of their behaviour (see Subsection 3.1.4). However, task locations remain uniformly distributed, and the discrete-time processes for announcement time and location are treated independently. In Chapter 7, we overcome these limitations using structured models of stochastic, continuous-time task announcement.

3.3.5 Simultaneous Task Allocation and Planning (STAP)

Task allocation and planning can be solved simultaneously for more effective coordination at the cost of increased complexity (see Fig. 3.1). In [232], robot paths and available tasks are shared globally in deterministic environments, similar to cooperative A* [67]. Robots then plan for the cheapest task they can execute without collisions. CBS is adapted for STAP in [233] to operate on multiple search trees, one for each allocation, where the first conflict-free solution contains the optimal allocation and plans. Similar to CBS, a hierarchical approach is taken in [234], which solves an abstract MRTA problem and refines it with hill climbing to consider robot planning. STAP can also be formulated as a mixed integer program, and solved through branch and bound techniques [235]. In [236], robots select temporal logic tasks and plan sequentially in a team model, where switch

transitions indicate a robot has finished task selection. This is extended to stochastic environments in [98]. In [103], robots choose tasks independently during planning by approximating the stochastic behaviour of the other robots. For partially observable environments, [237] models task execution with mixed observability MDPs (MOMDPs), and captures the MOMDP solutions in a factor graph for decentralised task allocation. Finally, for chance constrained STAP under stochastic durations, [238] iteratively solves a series of deterministic problems constructed from the duration means, variance, and a variable risk parameter. Similar to combined planning and model checking techniques (see Subsection 3.2.3), we do not consider STAP approaches to **Q3 - Planning** and **Q5 - Task Allocation** due to the complexity of solving the problems jointly, which would limit the scalability of our methods.

4

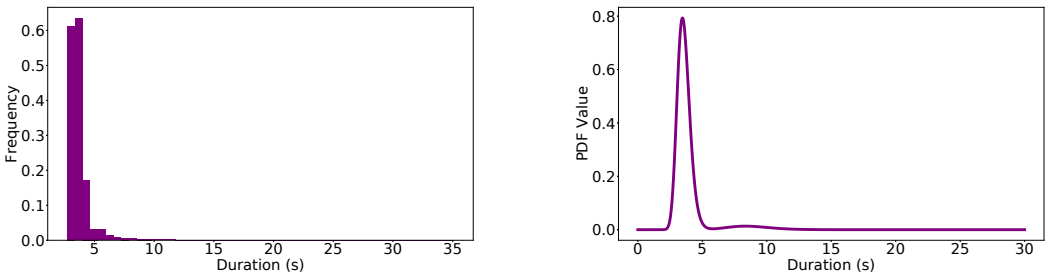
Phenomena that Affect Robot Navigation Durations

Contents

4.1 Introduction 51
4.2 The Effects of Congestion 53
4.2.1 Map Resource Groups 56
4.3 Contexts 58
4.3.1 Describing the Context for Multi-Robot Systems 59
4.3.2 Interaction Bands 61
4.4 Constructing Duration Models from Empirical Data . 62
4.5 Summary 64

4.1 Introduction

In this thesis, we consider multi-robot navigation tasks on topological maps. Many discrete-time multi-robot models such as MMDPs and MAPF models assume navigation actions have deterministic durations [17, 18]. This assumption fails to capture real execution-time robot behaviour, where navigation durations are affected by tire slip, localisation error etc. To synthesise robust multi-robot behaviour, and answer the questions set out in Chapter 1, we must model the temporal disturbances which affect the duration of navigation actions during execution. These disturbances



(a) A histogram of the observed data. **(b)** A distribution fit from the data.

Figure 4.1: Duration data for a simulated robot navigating on a topological edge, which demonstrates temporal uncertainty. Data was collected using the techniques in Section 4.4.

are difficult to predict during planning. For example, it is difficult to predict when and where unexpected obstacles may appear in the environment, which can cause robot localisation error. However, temporal disturbances can be represented implicitly as uncertainty over navigation duration. Further, in the real world, robot action durations are continuous, where the same continuous duration is never observed twice. We demonstrate this in Fig. 4.1a, which shows duration data for a mobile robot navigating along a topological edge. Though we often observe durations of around three seconds, the largest observed duration exceeds 15 seconds. Failure to capture this uncertainty leads to modelling errors which compound over multiple actions, leading to poor execution-time performance, as robots may choose routes likely to incur heavy delays for example. Therefore, in this thesis we use continuous duration distributions to capture the complex temporal dynamics of robot navigation. In Fig. 4.1b, we show a distribution fit from the data in Fig. 4.1a.

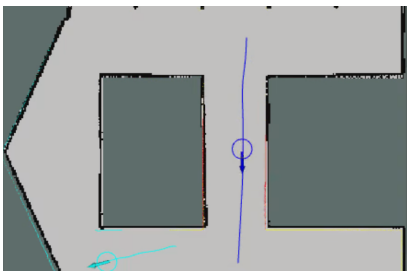
In this chapter, we construct probabilistic duration models for multiple mobile robots navigating on a topological map. These models are fundamental to the multi-robot modelling approach in Chapter 5, and the planning and task allocation methods in Chapters 6 and 7 respectively. We introduce *congestion*, a mode of interaction for mobile robots which affects robot navigation performance, and capture spatial dependencies between map resources. Further, we present *contexts* as a representation of the spatiotemporal situation robot actions are executed in, and augment topological edges with context-dependent duration distributions,

which we represent as PTDs (see Section 2.4). Contexts abstract the system state to capture the relevant information for identifying the duration profile of robot actions. We define two general context descriptors for MRSs which describe the presence of other robots near a robot's action location, which causes interactions such as congestion. With this, in Chapters 5 and 6 we use the context to identify congestion at the instant an action is triggered. We also capture similarities between contexts, and discuss approaches for collecting navigation duration data in a robotic simulator that trade between data efficiency and accuracy, which we use to build duration models which are representative of real robot execution. Though we focus on topological navigation, the techniques in this chapter are applicable to arbitrary robot actions. This chapter combines work which has appeared in [23–25].

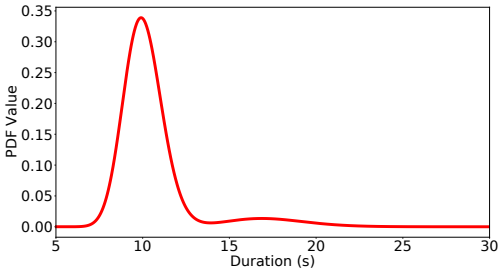
4.2 The Effects of Congestion

Congestion is a mode of interaction for mobile robots which occurs when multiple robots operate in the same area simultaneously. When congestion occurs, robots must manoeuvre around each other to prevent collisions. In this section, we analyse how congestion affects robot navigation performance, addressing **Q1 - Interactions**. We describe the level of congestion by counting the number of other robots present on a topological edge when a new robot begins to navigate along it. A robot is present at an edge $e = (v, v')$ if they are traversing it, i.e. they are travelling between v and v', and present at a node v if they are stationary at v, or executing an action there. Initially, for a robot navigating on edge (v, v'), we count the robots on edges (v, v') and (v', v), as they share the same physical space. We expand on this in Subsection 4.2.1 to capture spatial dependencies between map resources, and present two alternative descriptions of congestion which consider these dependencies in Subsection 4.3.1.

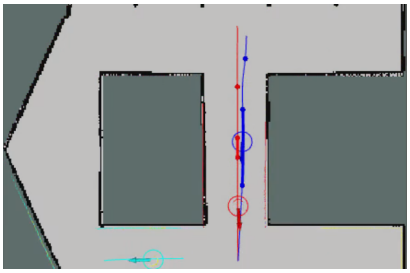
In Fig. 4.2, we demonstrate how congestion affects navigation duration. We focus on a single robot simulated in Stage [239], shown in dark blue, which navigates through a tunnel under different levels of congestion. We construct a duration distribution for each level of congestion from empirical data, which we discuss



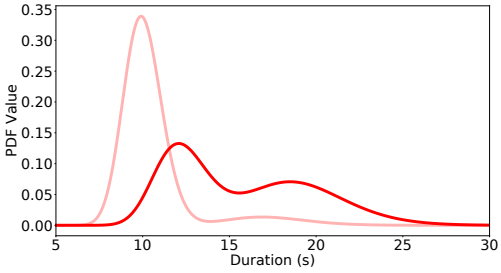
(a) The dark blue robot navigates through the uncongested tunnel.



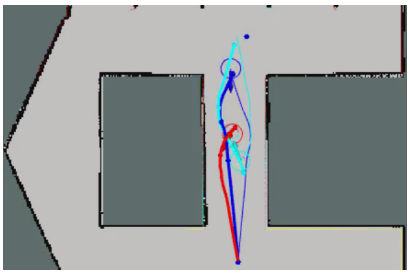
(b) The navigation duration distribution when the tunnel is uncongested.



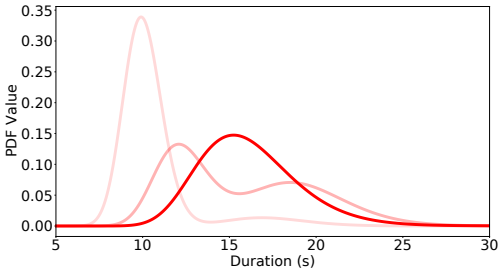
(c) The dark blue robot navigates through the tunnel when one other robot is present.



(d) The navigation duration distribution when one other robot is present in the tunnel.



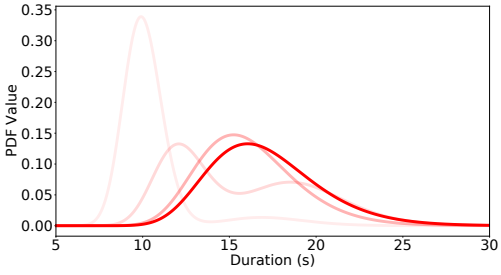
(e) The dark blue robot navigates through the tunnel when two other robots are present.



(f) The navigation duration distribution when two other robots are present in the tunnel.



(g) The dark blue robot navigates through the tunnel when three other robots are present.



(h) The navigation duration distribution when three other robots are present in the tunnel.

Figure 4.2: A demonstration of how congestion affects navigation duration. The circles with arrows show robots in the map, and coloured lines are possible robot paths.

further in Section 4.4. Note that the navigation duration for a given congestion level is stochastic, i.e. the number of other robots on an edge are not enough to deterministically predict the edge duration. When there are zero other robots in the tunnel (see Fig. 4.2a and Fig. 4.2b), robot navigation is only affected by minor disturbances such as tire slip, or variations in motor speed due to battery levels. Therefore, the duration distribution has low variance, where the mean duration is close to the minimum the robot can achieve given its speed.

In Fig. 4.2c, the red robot is in the tunnel when the dark blue robot tries to navigate. As the red robot starts to leave the tunnel it slows down, and the dark blue robot must slow down with it to avoid collisions. This increases navigation duration, as seen in Fig. 4.2d, where the distribution has two modes. Each mode likely represents a different direction of travel for the other robot. For example, if both robots travel in the same direction, the new robot must wait, as in Fig. 4.2c. However, if the robots travel in opposite directions, they must manoeuvre around each other, which has a different effect on navigation duration. By summing the robots in either direction, we combine multiple congestion scenarios. To address this, in Subsection 4.3.1 we consider a more precise method of detecting the congestion scenario which considers the direction of travel, and robots on different map resources.

In Fig. 4.2e and Fig. 4.2g, the dark blue robot traverses the tunnel when two and three other robots are present respectively. Navigation duration increases with congestion, as collision avoidance manoeuvres become more difficult, and robots must often wait until the number of other robots decreases. For example, in Fig. 4.2g the purple robot moves to the right to avoid the light blue robot, blocking the tunnel for the dark blue robot and preventing navigation. Moreover, congestion typically increases duration uncertainty, as for larger numbers of robots there is increased variation over how congestion occurs. For example, robots may travel in the same or opposite direction, and appear at different points on the edge whenever a new robot enters it; the examples in Fig. 4.2e and Fig. 4.2g show only one possibility. Each of these scenarios requires a different manoeuvre with a different duration.

Note there is little difference in congestion effects for two and three other robots (see Fig. 4.2f and Fig. 4.2h). One cause of this is found in Fig. 4.2g, where the red robot leaves the edge before the dark blue robot enters its proximity, removing the need for avoidance manoeuvres. We capture this in Subsection 4.3.2 by considering congestion situations with similar duration effects together.

In summary, congestion affects the duration of mobile robot navigation actions. The effects of congestion increase with the number of robots operating in the same location, i.e. robot navigation durations increase and become more uncertain. In Section 4.3, we introduce contexts to describe the spatiotemporal situation robot actions are executed in. We use contexts to describe the presence of other robots in the environment (see Subsection 4.3.1), which allows us to detect when and where congestion may occur.

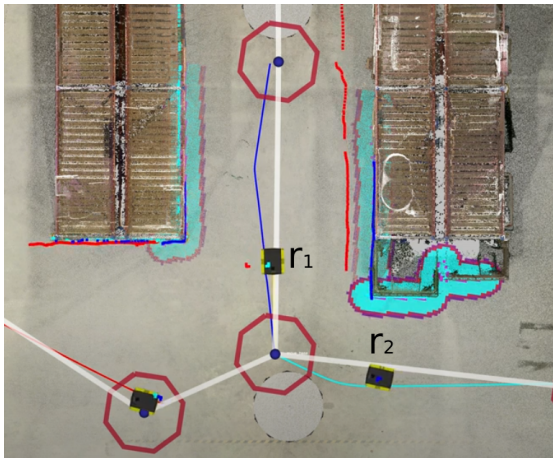
4.2.1 Map Resource Groups

During execution, robot interactions such as congestion can occur between robots at different, but nearby, map resources.

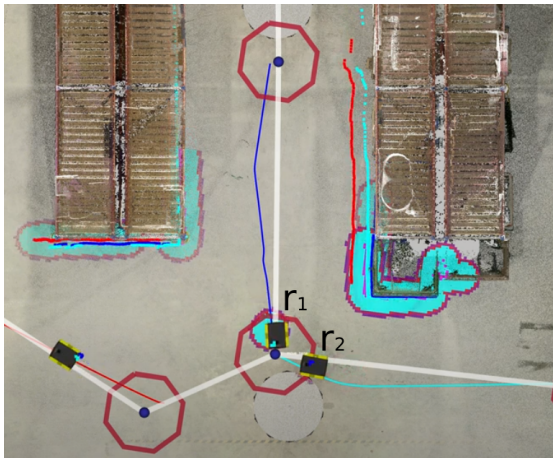
Example 4.1. Consider Fig. 4.3, which shows robots navigating in a warehouse. In Fig. 4.3a, robots r_1 and r_2 are navigating towards the same topological node from separate edges. Both robots approach the node at similar times, but r_1 is slightly earlier (see Fig. 4.3b). With this, r_1 moves into the node to complete its navigation action, which temporarily blocks r_2, slowing it down (see Fig. 4.3c). Robot r_1 eventually leaves the node to execute its next action, which allows robot r_2 to finish traversing the edge (see Fig. 4.3c).

We represent spatial dependencies between map resources using *map resource groups*, which we use in Subsection 4.3.1 to capture robot presence in locations that affect action execution.

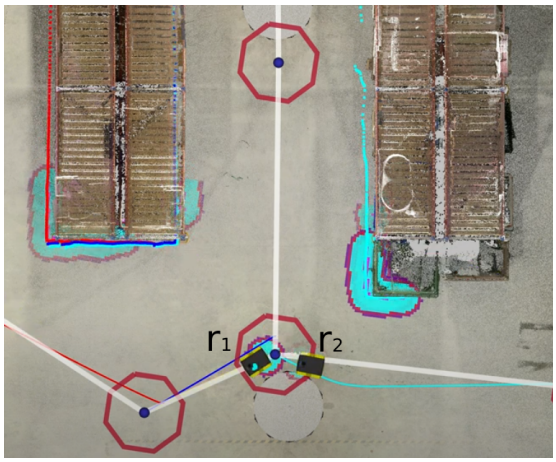
Definition 4.1. A *map resource group function* $g : X \to 2^X$ maps each map resource x to the set of map resources that affect robot action execution at x.



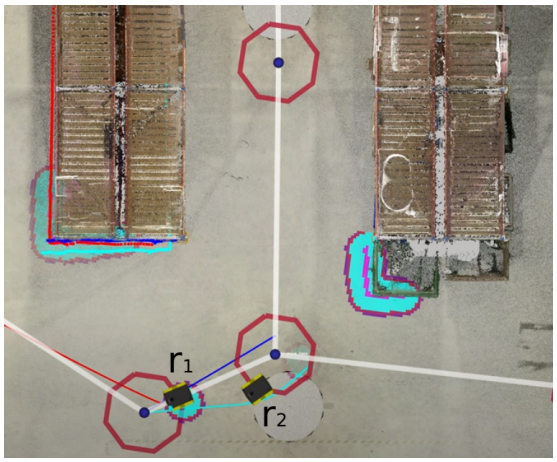
(a) Robots r_1 and r_2 are navigating on separate edges.



(b) The robots approach the same node at similar times.



(c) Robot r_1 blocks r_2 at the node, preventing navigation.



(d) Congestion is resolved and both robots continue navigating.

Figure 4.3: An example demonstrating spatial dependencies between map resources. Blue circles are topological nodes, and white lines are edges.

Example 4.2. Consider Fig. 4.4, which shows an abstracted version of the junction in Fig. 4.3, where robot r_1 is about to traverse (v_1, v_3). The presence of robot r_2 on (v_3, v_1) affects the duration of traversing (v_1, v_3), as the edges share the same physical space (see Fig. 4.2). Moreover, the presence of robot r_3 on (v_2, v_3) affects r_1, as the robots may block each other upon reaching v_3 (see Fig. 4.3). The same applies to any robots present on (v_4, v_3), or at v_3 itself. Therefore, $g((v_1, v_3)) = \{(v_1, v_3), (v_3, v_1), (v_2, v_3), (v_4, v_3), v_3\}$.

For all experiments in this thesis, we manually select map resource groups based on observations such as Example 4.1. However, map resource groups could be

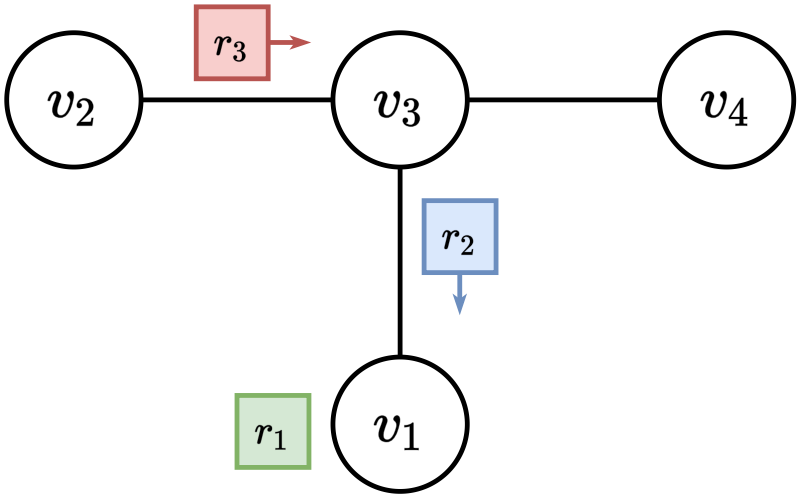


Figure 4.4: An MRS acting on a topological map, where squares represent robots. Robot r_1 is about to navigate along edge (v_1, v_3); robots r_2 and r_3 are navigating along (v_3, v_1) and (v_2, v_3) respectively.

learned automatically. For example, we could collect a dataset for each edge (see Section 4.4) which records the number of robots on every map resource whenever a robot begins traversing that edge, along with the duration. Feature selection techniques [240] could then be applied to this data to identify which map resources have the largest effect on the edge duration.

4.3 Contexts

The outcome and duration of robot actions are dependent on the precise spatiotemporal situation in which they are executed, which we refer to as the *context*. In this section, we formalise contexts for robots acting on topological maps, and present two general context descriptors for MRSs which we later use to detect when and where congestion may occur. Recall that map resources $x \in X$ are the nodes and edges of a topological map $\mathcal{T}$, $X = E \cup V$. Actions are executed at map resources, and we define contexts at each map resource $x \in X$, which capture the state of the robots and environment at the instant an action is triggered.

Definition 4.2. The complete set of *contexts* is given by $C = \bigcup_{x \in X} C_x$, where C_x denotes the set of contexts observable at map resource $x \in X$.

Contexts are an abstraction of the system state which capture the relevant features for identifying the duration of robot actions. To support topological navigation, we represent the environment as a *context-aware topological map*, where context-dependent duration distributions capture edge traversal. By identifying the context we select duration profiles which more accurately capture robot dynamics. In Section 4.4, we describe how to construct these distributions from empirical data.

Definition 4.3. A *context-aware topological map* is a tuple $\mathcal{T}_C = \langle V, E, C, \rho_E \rangle$, where C is the set of contexts, and $\rho_E : E \times C \to \text{Dist}(\mathbb{R}_{\geq 0})$ maps an edge and context for that edge to a distribution over the duration for a robot to traverse that edge.

4.3.1 Describing the Context for Multi-Robot Systems

Contexts capture phenomena which affect robot action execution, which are problem dependent. For example, if a robot is navigating indoors, its actions are unaffected by the weather. In this subsection, we present two context description methods for MRSs which use map resource groups to capture robot presence at locations that affect action execution. For this, let $\texttt{cnt}_i(x, t)$ return the number of other robots at map resource $x \in X$ at time t, from the perspective of robot r_i. Further, let $c_i(x, t)$ return the context observed by robot r_i immediately before executing an action at map resource x at time t.

Scalar Contexts

Let n be the total number of robots. Scalar contexts describe the context with a single number of other robots, $C_x = [\![0, n-1]\!], \forall x \in X$. If robot r_i executes an action at map resource $x \in X$, we observe the context by counting the number of other robots present anywhere in the map resource group for x. Formally:

$$c_i(x, t) = \sum_{x' \in g(x)} \texttt{cnt}_i(x', t). \tag{4.1}$$

By not distinguishing between map resources, scalar contexts aggregate multiple robot configurations within the map resource group. For example, a scalar context of three can represent three robots on the same edge or different edges. Each of

these configurations may have different duration effects, as the effects of congestion change depending on the direction of travel for example (see Section 4.2). Therefore, we consider vectorising the context to improve precision.

Vector Contexts

For actions executed at map resource $x \in X$, vector contexts describe the context as a set of (resource, #robots) pairs for each location in the map resource group for x, $C_x = 2^{X \times [\![0, n-1]\!]}$, $\forall x \in X$. Formally:

$$c_i(x, t) = \{(x', \texttt{cnt}_i(x', t)) \mid x' \in g(x)\}. \tag{4.2}$$

Vector contexts count for each map resource separately, which distinguishes between robot configurations more precisely than scalar contexts, but increases the number of contexts. In Section 5.5, we compare how scalar and vector contexts impact models of multi-robot behaviour when fitting context-dependent duration models from empirical data (see Section 4.4).

Example 4.3. In Fig. 4.4, robot r_1 is about to navigate edge (v_1, v_3), and we want to observe the context. In this example, let the map resource group for (v_1, v_3) be $g((v_1, v_3)) = \{(v_1, v_3), (v_3, v_1), (v_2, v_3), (v_4, v_3), v_3\}$. Since r_2 is present on (v_3, v_1), and r_3 on (v_2, v_3), the scalar context is 2, i.e. there are two robots. The vector context is $(((v_1, v_3), 0), ((v_3, v_1), 1), ((v_2, v_3), 1), ((v_4, v_3), 0), (v_3, 0))$.

Scalar and vector contexts describe the presence of other robots near a robot's action location. As more robots operate in the same area simultaneously, the effects of congestion increase, which affects the duration of robot navigation actions (see Section 4.2). By identifying the scalar or vector context an action is executed under, we can identify when and where congestion will occur, and the effect it has on navigation duration. In Chapters 5 and 6, we compute the scalar or vector context at the instant an action is triggered, and select the corresponding duration distribution for that context, which captures the effects of congestion. Each congestion scenario a robot may experience during execution maps to a single context which has a

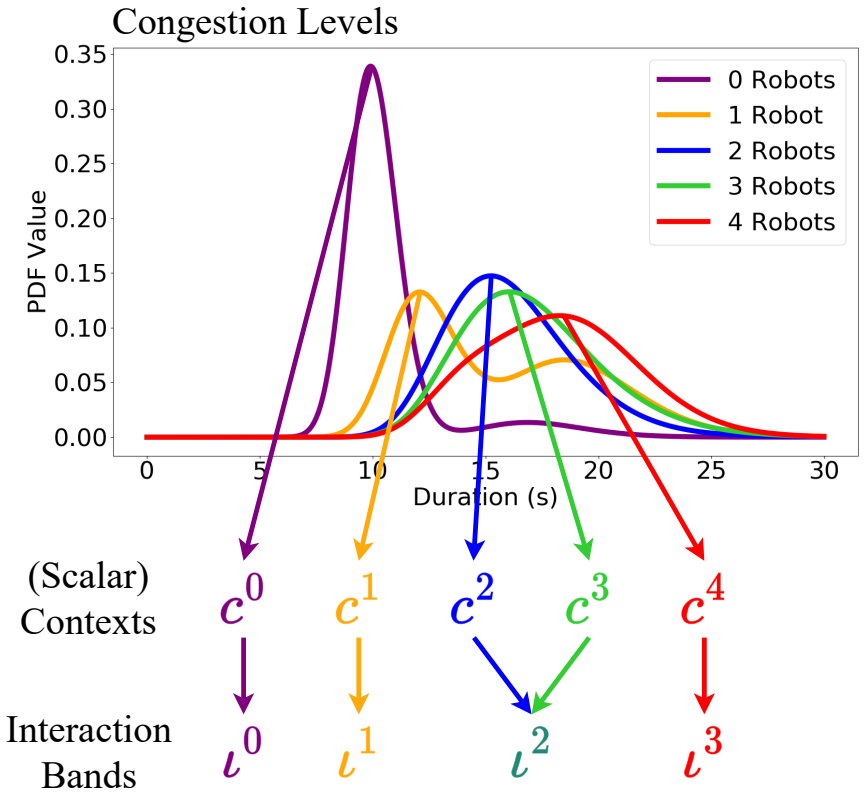


Figure 4.5: The relationships between congestion, contexts, and interaction bands.

corresponding duration distribution, as seen in Fig. 4.5. In Chapter 5, we reformulate scalar and vector contexts over the joint state of a multi-robot MA. Scalar and vector contexts assume homogeneous robots, i.e. each robot affects action duration similarly. For heterogeneous MRSs composed of different classes of robot, we can extend the notion of scalar and vector contexts to count how many robots of *each class* are at a map resource, rather than the total number of robots.

4.3.2 Interaction Bands

During execution, similar duration effects are observed under different contexts. For example, consider Fig. 4.5, which includes the distributions in Fig. 4.2, where for scalar contexts c^2 and c^3, there is little difference in the duration distributions. To capture similarities in duration effects, we partition the contexts at each map resource into *interaction bands*. Formally:

Definition 4.4. A set of *interaction bands* at a map resource $x \in X$ is a partition over the context set C_x, given by $I_x = \{\iota_x^0, \iota_x^1, ..., \iota_x^b\}$, where $\iota_x^j \subseteq C_x$ is a subset of the contexts at x, and each context appears in exactly one interaction band, i.e. $\forall j \neq k$, $\iota_x^j \cap \iota_x^k = \emptyset$, and $\bigcup_j \iota_x^j = C_x$.

To simplify notation, we omit subscript x from interaction band ι_x^j if x can be inferred from the context. Further, we write $I = \bigcup_{x \in X} I_x$ to denote the complete set of interaction bands. In Fig. 4.5, scalar contexts c^2 and c^3 are grouped into an interaction band ι^2. Contexts in the same interaction band are assumed to have similar duration effects. Therefore, ρ_E returns the same distribution for all contexts in a given band, i.e. for edge $e \in E$ and band ι^j, $\rho_E(e, c) = \rho_E(e, c')$, $\forall c, c' \in \iota^j$. In Section 4.4, we fit duration distributions from empirical data. The data for contexts in the same interaction band is aggregated to produce a single distribution. We use interaction bands in Chapter 6 to improve the scalability of planning, and in Section 6.8 we analyse the effect interaction bands have on the accuracy of our duration models. Similar to map resource groups, we manually select the interaction bands for all experiments in this thesis. However, interaction bands could be learned automatically by measuring the similarity between the empirical distributions for each context [241].

4.4 Constructing Duration Models from Empirical Data

To capture robot navigation and interactions such as congestion, we require probabilistic duration models that are representative of real robot execution. In this section, we achieve this by fitting continuous duration distributions from empirical navigation data, which can be collected in the real world or physics-based simulators such as Gazebo [155] and Stage [239].

To collect duration data, we design navigation behaviours which the robots execute. Whenever a robot traverses an edge, the duration and context are recorded. We record the context when a robot starts an edge to match our modelling

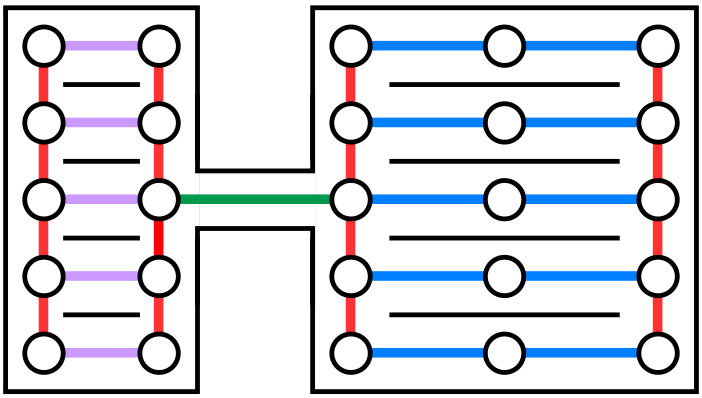
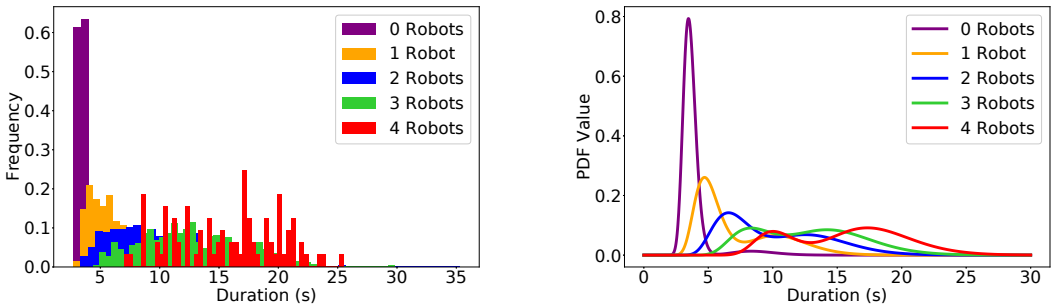


Figure 4.6: An example edge partition, where edges with the same colour are of the same length.



(a) Histograms of the collected data. **(b)** PTDs fit from the data in Fig. 4.7a.

Figure 4.7: Example PTDs for different scalar contexts on a topological edge, where the PTDs are fit from empirical duration data.

assumptions in Chapters 5 and 6. We present two behaviour methods for data collection. The first is random navigation across the topological map. For the second, we exploit regularity within robotic environments, such as the aisles of a warehouse, by partitioning edges based on length, where equal length edges appear in the same subset (see Fig. 4.6). We then target data collection on one edge from each subset by sending robots on routes containing the target edge, which causes interactions to occur, and allows us to observe contexts with higher numbers of robots. The data for a target edge is used for all edges in its subset, and so edge partitioning ignores differences between equal length edges, such as the floor surface or physical surroundings. In Chapter 5, we assess how the data collection methods impact the accuracy of our duration models. We present edge partitioning as a straightforward approach for increasing the speed of data collection by focusing on fewer edges.

Random navigation naively collects data for all edges, and is used as a comparison to edge partitioning in Chapter 5. More informed data collection methods are possible, but are not a focus of this work. For example, we could use similarity measures to identify edges with similar durations [241], and share data between those edges.

From navigation data, we fit duration distributions for each topological edge and context/interaction band, as required by ρ_E in Section 4.3. We represent the duration distributions with PTDs (see Section 2.4), which allow for arbitrary distribution shapes, and fit them using the method in [51], as described in Appendix D. In Fig. 4.7, we show an example set of PTDs fit from empirical data. Context-dependent PTDs are fundamental to our approach for **Q2 - Modelling**, allowing for the construction of multi-robot MA in Chapter 5, and the interpretation of robot policies as CTMCs in Chapter 6. As well as collecting duration data, we also observe the probability of successful edge navigation for each context by counting the number of successes and failures, which we add to our multi-robot model in Chapter 5.

4.5 Summary

In this chapter, we constructed probabilistic duration models which capture temporal uncertainty and the effects of robot interactions. To address **Q1 - Interactions** we analysed how congestion affects robot navigation performance, where navigation durations typically increase and become more uncertain as more robots are present in the same location simultaneously. In Chapter 6 we reason over the effects of congestion to synthesise efficient behaviour for multi-robot navigation tasks. Further, we introduced the notion of contexts, which abstract the state of the system and capture the spatiotemporal situation robot actions are executed in. With this, we defined scalar and vector contexts, which describe the context for MRSs in terms of the presence of other robots near a robot's action location. Each scalar and vector context maps to a duration distribution which captures the effects of robot interactions such as congestion under that context. To construct duration models which are representative of real robot execution, we also presented techniques for collecting navigation duration data. In Chapter 5, we analyse how the context

description methods and data collection techniques affect our models of multi-robot behaviour. Probabilistic duration models form the core of our approach to **Q2 - Modelling**, presented in subsequent chapters. The work in this chapter appears in multiple publications. Congestion and interaction bands were first introduced in [23], and discussed further in [24]. Contexts, map resource groups, and the data collection techniques in Section 4.4 first appeared in [25].

5

Context-Aware Modelling for Multi-Robot Systems Under Uncertainty

Contents

5.1 Introduction 67
5.2 Context-Aware Multi-Robot MA 69
5.2.1 MRS Assumptions 69
5.2.2 Single-Robot MA 71
5.2.3 Multi-Robot MA 74
5.3 Context-Aware Multi-Robot Simulation 75
5.3.1 Robot Policies 76
5.3.2 The CAMAS Algorithm 78
5.4 Complexity Analysis 80
5.4.1 MRMA Space Complexity 80
5.4.2 CAMAS Time Complexity 81
5.5 Experiments 81
5.5.1 Experimental Setup 81
5.5.2 Evaluating Duration-Based Metrics 85
5.5.3 CAMAS Simulation Speed 89
5.5.4 Evaluating Failure-Based Metrics 89
5.6 Discussion 93
5.7 Summary 94

5.1 Introduction

To address **Q2 - Modelling** and support coordination, we require a formal model of the MRS. Formal multi-robot models capture the task-level behaviour of the MRS abstract of low-level components such as controllers, i.e. they consider high-level capabilities such as navigation or manipulation, while omitting how the robot achieves this during execution. Multi-robot planning techniques [19] reason over formal models to synthesise robot behaviour, whereas simulation [163] and model checking [176] techniques use formal models alongside synthesised behaviour to evaluate task-level metrics of multi-robot performance (see Chapter 3 and Fig. 3.1). The success of these techniques is limited by the accuracy of the multi-robot model [164]. For example, if we plan on an inaccurate model, our expectations of robot behaviour during planning diverge from what we observe during execution. For the problems in this thesis, an inaccurate model could cause robots to experience unexpected delays during execution. If the robots are not aware of these delays, they cannot plan to mitigate their effects, decreasing performance. As discussed in Chapter 3, existing multi-robot models make strong assumptions, such as simplistic duration models, e.g. deterministic or Gaussian models [6, 103]; robot independence [242]; and synchronised execution [17]. These assumptions fail to correlate with our observations of execution, decreasing the predictive power of our models. For example, by assuming robot independence, we ignore execution-time interactions such as congestion, which can cause significant robot delays. To accurately capture high-level multi-robot behaviour, we require formal models that capture uncertain action durations and robot interactions in continuous time.

In this chapter, we address **Q2 - Modelling** by presenting a multi-robot Markov automaton (MRMA) formulation that models asynchronous multi-robot execution in continuous time; see Section 2.5 for an overview of MA. We abstract robot dynamics using the context-dependent duration distributions presented in Chapter 4, which are represented as PTDs fit from empirical data [49] on a context-aware topological map. Further, we adapt scalar and vector contexts to the MRMA, and use them to detect robot interactions such as congestion at the instant actions are triggered. Combining

the MRMA structure with PTDs built from empirical data enables our model to capture the behaviour of robots interacting to achieve tasks in a physical world.

The MRMA has numerous applications. First, the MRMA can be used for multi-robot planning under uncertain durations and interactions. The MRMA can also be used for multi-agent reinforcement learning [243], where the MRMA could act as a model-based surrogate to speed up learning while maintaining solution quality. Moreover, the MRMA could be used as a discrete-event simulator (DES) to simulate high-level multi-robot behaviour while avoiding the computational overhead of more realistic physics-based simulators [161]. We present a scalable DES called CAMAS (context-aware multi-agent simulator), which addresses **Q4 - Policy Evaluation** by sampling through the MRMA under a set of robot policies to evaluate task-level properties of multi-robot behaviour, such as the time to complete a set of tasks, or the number of robot failures. We use CAMAS to empirically validate the MRMA, demonstrating that our model accurately captures the behaviour of an MRS simulated in Gazebo.

In this chapter, we use MAs instead of similar continuous-time models such as CTMDPS [117] and GSMDPs [52] (see Subsection 3.1.6). CTMDPs represent action durations with a single exponential distribution, which fails to capture real robot execution (see Chapter 4). Using MAs, we capture instantaneous robot action choice with immediate transitions, and action durations with PTDs, i.e. sequences of exponential transitions, which can model arbitrary distribution shapes [49] (see Section 2.4). GSMDPs allow for arbitrary duration distributions but are complex to define and solve, where standard GSMDP solutions convert duration distributions into PTDs [52] (see Subsection 3.1.6). This converted model is similar to MAs, which can be solved directly. There also exists model checking techniques for MAs [57], which widens the possible applications of our models. Further, CAMAS is easier to construct over an MRMA than a GSMDP, as the memoryless property of the exponential distribution removes the need to track how long has passed since the start of an action. The work in this chapter has been published in [25].

5.2 Context-Aware Multi-Robot MA

In this section, we present an MRMA that uses contexts to identify when and where robot interactions such as congestion will occur, where stochastic action durations are represented with PTDs.

5.2.1 MRS Assumptions

Prior to building the MRMA in Subsection 5.2.3, we first list our assumptions over the joint MRS state space and robot action execution, given a team of n robots $R = \{r_1, ..., r_n\}$.

- We assume the environment is represented as a context-aware topological map $\mathcal{T}_C = \langle V, E, C, \rho_E \rangle$, where all task locations must have a corresponding topological node. Topological maps are commonly used in task-level models of robot execution [34, 236, 244], but contexts are a novel formalism introduced in this thesis (see Section 4.3). Robot r_i can navigate on the topological map and execute a set of non-navigation actions N_i at topological nodes, such as grasping an object or opening a door. We denote the actions r_i can execute with $A_i = (E \cup N_i)$. The complete set of non-navigation actions is denoted $N = \bigcup_i N_i$. Non-navigation action durations are defined by $\rho_N : N \times C \to \mathbb{P}$, where $\rho_N(a, c)$ returns a PTD over the duration of executing action a under context c.

- We assume that the joint system state space can be factored into state features $S_J = S_{l,1} \times ... \times S_{l,n} \times S_g$. Here, $S_{l,i} = S^1_{l,i} \times ... \times S^{k_i}_{l,i}$ represents r_i's local state space, where $S^j_{l,i}$ is a local state feature such as the robot's battery level. We assume $S^1_{l,i} = V$, which we shorten to $S_{v,i}$, i.e. a robot's local state contains its location. Moreover, S_g is a possibly empty set of global state features shared among robots, such as whether doors are open, or the wall clock time, which we denote as $S_g = S^1_g \times ... \times S^m_g$. Factored state spaces are a common modelling approach which make many problems easier to specify without restricting the problems we can model, as any state space can be captured with

a single state feature [17, 37]. For certain classes of problems, factored state spaces can be exploited to improve the scalability of solution methods [245].

- To simplify notation we assume actions are deterministic. However, our MRMA formalism in Subsection 5.2.3 allows for stochastic action outcomes, as demonstrated in Section 5.5. The extension to stochastic outcomes is straightforward, and described in Subsection 5.2.2.

- We assume actions can only change the robot's local state and the global state features. We define the effect(s) of an action with $\mathit{eff}_i : S_{l,i} \times S_g \times A_i \rightarrow S_{l,i} \times S_g$, where $\mathit{eff}_i(s_{l,i}, s_g, a) = (s'_{l,i}, s'_g)$ returns the new local state for robot r_i and the updated global state features after r_i finishes executing action a. This assumption is used to prevent unrealistic behaviour during asynchronous multi-robot execution, such as robot r_1's action changing r_2's location.

- We assume that the context is sufficient to identify the duration profile of robot actions, and that the context does not change while an action is being executed, such that we can observe the context at the instant a robot begins action execution. These assumptions are specific to the MRMA we present in this chapter, and in Section 5.5 we evaluate their impact on model accuracy.

The above assumptions are less restrictive than for common multi-robot models such as MMDPs [17] and deterministic MAPF models [18]. First, we allow asynchronous robot execution, which avoids unnecessary waiting during execution [22] (see Subsection 3.1.3). Second, we do not assume that robot action durations are fixed and identical. Instead, we represent action durations as continuous stochastic processes for each context, which capture the effects of temporal uncertainty and robot interactions such as congestion. Under our assumptions, we can model any multi-robot problem where robots navigate to fixed locations to execute actions, and where robot interactions affect continuous-time robot behaviour. For example, we can model robots operating in a fulfilment centre, where each product is located near a topological node. Here, navigation actions only affect a robot's location, and

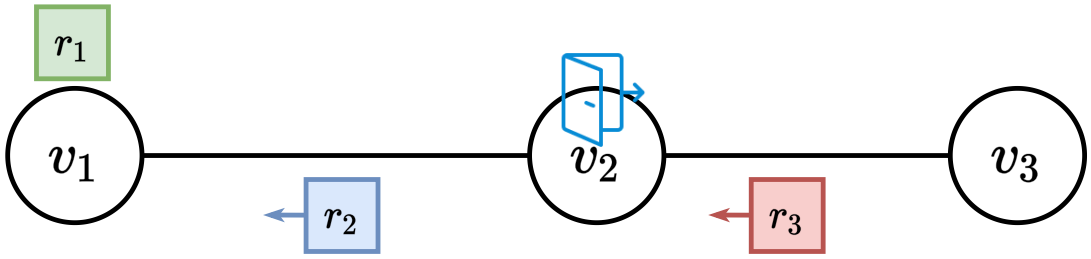


Figure 5.1: An MRS on a topological map, where squares represent robots and there is a door at v_2. Robot r_1 is stationary and about to navigate along topological edge (v_1, v_2).

picking actions affect the stock levels, which are a global state feature. Alternatively, we can model search and rescue problems, where there are nodes near each search location, and the status of survivors are represented with global state features.

Example 5.1. Consider Fig. 5.1, where robot r_i's local state contains the last node it arrived at and its battery level b_i. There is a closed door at v_2, represented in S_g as `open` or `closed`. The current joint state is $((v_1, b_1), (v_2, b_2), (v_3, b_3), \texttt{closed}) \in S_J$. Upon reaching v_2, robot r_3 opens the door. The effects of opening the door are $\mathit{eff}_3((v_2, b_3), \texttt{closed}, \texttt{open_door}) = ((v_2, b_3 - \gamma), \texttt{open})$, i.e. r_3's battery is decreased by γ, and the door is now open. After opening the door, the joint state is $((v_1, b_1), (v_2, b_2), (v_2, b_3 - \gamma), \texttt{open}) \in S_J$.

5.2.2 Single-Robot MA

In this subsection we describe how to build a single-robot MA $\mathcal{A}_i$ for each robot r_i, which we compose to build the MRMA in Subsection 5.2.3. In $\mathcal{A}_i$ (see Def. 5.1), robot r_i chooses an action and follows exponential transitions corresponding to a PTD for that action that assumes no other robots are present.

Definition 5.1. A *single-robot MA* is a tuple $\mathcal{A}_i = \langle S_{\mathcal{A}_i}, \bar{s}_{\mathcal{A}_i}, A_{\mathcal{A}_i}, \delta_{\mathcal{A}_i}, \Delta_{\mathcal{A}_i} \rangle$, where $S_{\mathcal{A}_i} = S_{l,i} \times S_{\mathbb{P}}^{\perp} \times S_g$, with $S_{\mathbb{P}}^{\perp} = S_{\mathbb{P}} \cup \{\perp\}$. A state $s = (s_{l,i}, s_{p,i}, s_g) \in S_{\mathcal{A}_i}$ comprises the robot's local state $s_{l,i}$, a PTD state $s_{p,i}$, and the global state feature values s_g (see Subsection 5.2.1). The PTD state tracks progress through the robot's current PTD, where $s_{p,i} = \perp$ indicates the robot is not executing an action. The initial state $\bar{s}_{\mathcal{A}_i} = (\bar{s}_{l,i}, \perp, \bar{s}_g)$ specifies the initial local and global state feature values, where the robot is idle. $A_{\mathcal{A}_i} = A_i$, where A_i is the action set for robot r_i as specified

in Subsection 5.2.1, i.e robots can navigate along edges and execute non-navigation actions. We write $A_i(s)$ to denote the enabled actions in state $s \in S_{\mathcal{A}_i}$.

Before defining the transition functions $\delta_{\mathcal{A}_i}$ and $\Delta_{\mathcal{A}_i}$, we require additional notation. For each action and context, there is a corresponding PTD, and each PTD maps to a single action (see Fig. 5.2a). Thus, for PTD state $s_{p,i}$, we write $\mathcal{P}(s_{p,i})$ to denote the PTD $s_{p,i}$ belongs to, and $a(s_{p,i})$ to denote the action modelled by $\mathcal{P}(s_{p,i})$; we define $a(s_{p,i}) = \perp$ when $s_{p,i} = \perp$. Further, we define a function $\texttt{PTD}_i : S_{\mathcal{A}_i} \times A_{\mathcal{A}_i} \to \mathbb{P}$, which returns the PTD followed by robot r_i when executing action a in state s:

$$\texttt{PTD}_i(s,a) = \begin{cases} \rho_E(a, c_i(s,a)) & \text{if } a \in A_i(s) \cap E \\ \rho_N(a, c_i(s,a)) & \text{if } a \in A_i(s) \cap N_i, \end{cases} \tag{5.1}$$

where c_i is the context function as in Subsection 4.3.1, but defined over states and actions, i.e. $c_i(s,a)$ is the context observed by r_i immediately before executing action a in state s. For $\mathcal{A}_i$, $c_i(s,a) = c^0, \forall s \in S_{\mathcal{A}_i}$ and $\forall a \in A_i(s)$, where c^0 denotes the absence of other robots. As $\mathcal{A}_i$ assumes r_i is the only robot, interactions such as congestion will never occur. When composing the single-robot models into the MRMA in Subsection 5.2.3, we will modify c_i to compute scalar and vector contexts over the joint state (see Subsection 4.3.1). Scalar and vector contexts describe the presence of other robots near robot r_i's action location, which allows us to identify whether interactions such as congestion will occur during action execution.

$\delta_{\mathcal{A}_i}$ models instantaneous robot action choice. Upon triggering action a in state s, the robot follows a sequence of exponential transitions which capture durative action execution, as defined by PTD $\texttt{PTD}_i(s,a)$ and exponential transition function $\Delta_{\mathcal{A}_i}$, discussed below. To begin action execution, transitions in $\delta_{\mathcal{A}_i}$ update the PTD state from $\perp$, which represents the robot being idle, to an initial state of $\texttt{PTD}_i(s,a)$, which represents the robot starting action a. The transition probability is defined by the PTD's initial state distribution (see Fig. 5.2). Formally, for $s = (s_{l,i}, s_{p,i}, s_g)$ and $s' = (s'_{l,i}, s'_{p,i}, s'_g)$:

$$\delta_{\mathcal{A}_i}(s,a,s') = \begin{cases} init_{\mathtt{PTD}_i(s,a)}(s'_{p,i}) & \text{if } s_{p,i} = \perp \text{ and} \\ & (s_{l,i}, s_g) = (s'_{l,i}, s'_g) \text{ and} \\ & a \in A_i(s) \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

$\Delta_{\mathcal{A}_i}$ captures the effects of the PTDs. If the robot is in PTD state $s_{p,i}$, the rate to a non-absorbing state $s'_{p,i} \in S_{\mathcal{P}(s_{p,i})}$ is defined according to $\Delta_{\mathcal{P}(s_{p,i})}$. When the robot finishes a PTD, its PTD state is set to $\perp$ and its local state, as well as the global state features, are updated. The rate of this transition is the rate to the absorbing state according to $\Delta_{\mathcal{P}(s_{p,i})}$. Formally, for $s = (s_{l,i}, s_{p,i}, s_g)$ and $s' = (s'_{l,i}, s'_{p,i}, s'_g)$:

$$\Delta_{\mathcal{A}_i}(s,s') = \begin{cases} \Delta_{\mathcal{P}(s_{p,i})}(s_{p,i}, s'_{p,i}) & \text{if } \mathcal{P}(s_{p,i}) = \mathcal{P}(s'_{p,i}) \text{ and} \\ & (s_{l,i}, s_g) = (s'_{l,i}, s'_g) \text{ and} \\ & s'_{p,i} \neq s^f_{\mathcal{P}(s'_{p,i})} \\ \Delta_{\mathcal{P}(s_{p,i})}(s_{p,i}, s^f_{\mathcal{P}(s_{p,i})}) & \text{if } s'_{p,i} = \perp \text{ and} \\ & \mathit{eff}_i(s_{l,i}, s_g, a(s_{p,i})) = (s'_{l,i}, s'_g) \\ 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

The transition functions can be extended to handle stochastic outcomes by using a PTD for each outcome, where upon action choice there are immediate transitions to the initial states of these PTDs, weighted by the outcome probability. Upon finishing a PTD, the state update is outcome dependent. We demonstrate this in Section 5.5, where we integrate terminal robot navigation failure into the MRMA. Stochastic outcomes increase the burden of data collection, as we require larger datasets to capture each outcome's duration.

Example 5.2. Consider robot r_1 in Fig. 5.1, where the current state in $\mathcal{A}_1$ is $s = ((v_1, b_1), \perp, \texttt{closed})$. The only enabled action for r_1 is $A_1(s) = \{(v_1, v_2)\}$, i.e. navigate along (v_1, v_2). Upon choosing action (v_1, v_2), r_1's state is updated to $((v_1, b_1), s_{p,1}, \texttt{closed})$, where $s_{p,1} \in S_{\rho_E((v_1,v_2),c^0)}$, as r_1 has begun to navigate. Robot r_1 then follows a sequence of exponential transitions, as defined by $\rho_E((v_1, v_2), c^0)$, until it arrives at v_2.

5.2.3 Multi-Robot MA

To model asynchronous multi-robot execution and interactions, in this subsection we describe how to build an MRMA $\mathcal{A}_J$ that models the state features in S_J (see Subsection 5.2.1) and each robot's PTD state by composing the single-robot MAs $\mathcal{A}_i$, $i \in [\![1, n]\!]$. Each transition in $\mathcal{A}_J$ models a robot's action choice or PTD progression, and upon action choice we compute the scalar and vector contexts over the joint state to detect robot interactions (see Subsection 4.3.1).

Definition 5.2. A *context-aware MRMA* is a tuple $\mathcal{A}_J = \langle S_{\mathcal{A}_J}, \bar{s}_{\mathcal{A}_J}, A_{\mathcal{A}_J}, \delta_{\mathcal{A}_J}, \Delta_{\mathcal{A}_J} \rangle$, where $S_{\mathcal{A}_J} = (\bigtimes_{i \in [\![1,n]\!]} (S_{l,i} \times S_{\mathbb{P}}^{\perp})) \times S_g$, i.e. each robot's local and PTD state, and the global state features. The initial state is $\bar{s}_{\mathcal{A}_J} = (\bar{s}_{l,i}, \perp, ..., \bar{s}_{l,n}, \perp, \bar{s}_g)$, i.e. each robot's initial local and PTD states, and the initial global state feature values. The action set $A_{\mathcal{A}_J} = \bigcup_i A_{\mathcal{A}_i}$ is the union of each robot's action set. Let $[\cdot]_i$ project states in $S_{\mathcal{A}_J}$ onto $S_{\mathcal{A}_i}$, i.e. for state $s = (s_{l,1}, s_{p,1}, ..., s_{l,n}, s_{p,n}, s_g)$, let $[s]_i = (s_{l,i}, s_{p,i}, s_g)$, and $[s]_{l,i} = s_{l,i}$. Immediate transitions model a single robot's action choice, which changes only their local state:

$$\delta_{\mathcal{A}_J}(s, a, s') = \begin{cases} \delta_{\mathcal{A}_i}([s]_i, a, [s']_i) & \text{if } a \in A_{\mathcal{A}_i} \text{ and} \\ & \forall j \neq i,\ [s]_j = [s']_j \\ 0 & \text{otherwise.} \end{cases} \tag{5.4}$$

Exponential transitions model a single robot's PTD progress, which may change only their local state and the global state features:

$$\Delta_{\mathcal{A}_J}(s, s') = \begin{cases} \Delta_{\mathcal{A}_i}([s]_i, [s']_i) & \text{if } \forall j \neq i,\ [s]_{l,j} = [s']_{l,j} \\ 0 & \text{otherwise.} \end{cases} \tag{5.5}$$

Recall from Subsection 5.2.2 that $c_i(s, a)$ is the context observed by robot r_i when executing action a in state s. In Eq. 5.1, c_i ignores the other robots, and fails to detect robot interactions such as congestion. We now redefine $c_i : S_{\mathcal{A}_J} \times A_{\mathcal{A}_J} \to C$ to compute scalar and vector contexts (see Subsection 4.3.1) over MRMA states, which we use to identify when and where robot interactions will occur. First, we

redefine the function $\texttt{cnt}_i$ in Subsection 4.3.1 to count how many other robots are at map resource x in joint state s:

$$\texttt{cnt}_i(s, x) = \sum_{k \in [\![1,n]\!] \setminus \{i\}} \mathbb{1}[L(s_{v,k}, a(s_{p,k})) = x]. \tag{5.6}$$

In the above, $\mathbb{1}$ is the indicator function and $L : V \times (E \cup N \cup \{\perp\}) \to X$ represents where an action is taking place, defined as $L(v, a) = a$ if $a = (v, v') \in E$, and $L(v, a) = v$ otherwise. In words, navigation actions take place on the edge being navigated, and non-navigation actions occur at the node the action is executed at. Robots that are idle in a node are also considered: if r_k is in a state where $s_{p,k}$ is such that $a(s_{p,k}) = \perp$, then $L(s_{v,k}, a(s_{p,k})) = s_{v,k}$. For both context definitions, we use function L to identify an action's location, and use $\texttt{cnt}_i$ to count the robots present in that location's map resource group. For scalar contexts, we modify Eq. 4.1 to sum the robots present anywhere in the action location's map resource group:

$$c_i(s, a) = \sum_{x \in g(L(s_{v,i}, a))} \texttt{cnt}_i(s, x), \tag{5.7}$$

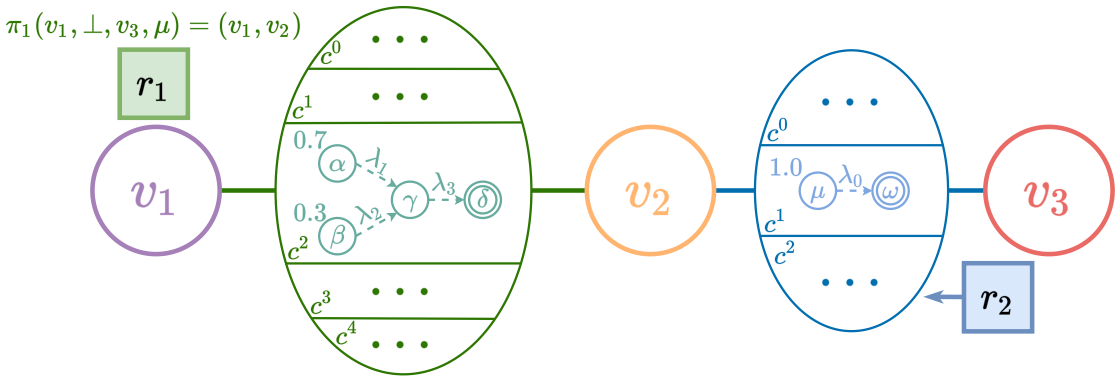
and for vector contexts, we modify Eq. 4.2 to compute a (resource, #robots) pair for each location in the action location's map resource group:

$$c_i(s, a) = \{(x, \texttt{cnt}_i(s, x)) \,|\, x \in g(L(s_{v,i}, a))\}. \tag{5.8}$$

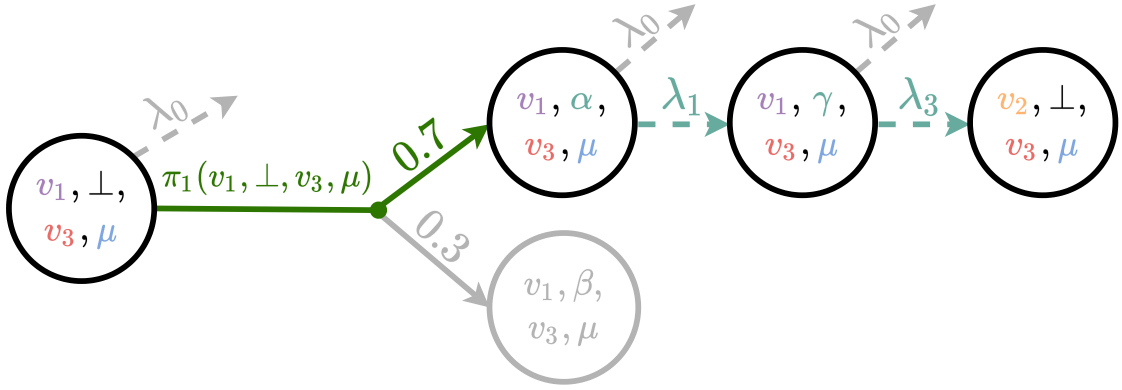
Example 5.3. Consider the MRS in Fig. 5.2a, where each robot's local state is its current node, $S_{l,i} = V$, and there are no global state features. Robot r_1 is stationary, and r_2 is navigating (v_3, v_2) under context c^1. The current MRMA state is $(v_1, \perp, v_3, \mu)$ (see Fig. 5.2b), i.e. r_1 is idle at v_1, and r_2 is in PTD state $\mu \in S_{\rho_E((v_3,v_2),c^1)}$, coming from v_3.

5.3 Context-Aware Multi-Robot Simulation

In this section, we introduce CAMAS, a DES which simulates asynchronous multi-robot behaviour in continuous time while capturing robot interactions such as congestion. Given a set of robot policies, CAMAS simulates multi-robot behaviour



(a) An MRS on a context-aware topological map visualising the leftmost state in Fig. 5.2b.



(b) The simulation run. Grey transitions are enabled but not sampled.

Figure 5.2: A simulation run through a 2 robot MRMA $\mathcal{A}_J$. Solid arrows represent immediate transitions, dashed arrows represent exponential transitions, and concentric circles represent PTD absorbing states.

by sampling execution traces through an MRMA. From statistics over these traces, we can evaluate **any** property of multi-robot behaviour, such as the sum of robot task durations, the number of tasks completed without failure, or satisfaction of temporal logic specifications. We begin in Subsection 5.3.1 by discussing robot policies, which specify robot behaviour in CAMAS. With this, in Subsection 5.3.2, we present the CAMAS algorithm for multi-robot policy evaluation, which addresses **Q4 - Policy Evaluation**.

5.3.1 Robot Policies

Prior to simulation, CAMAS receives a policy π_i for each robot r_i which defines their behaviour. In this subsection, we describe the different types of policy supported

by CAMAS. For a fully observable state space S, and action set A, a policy π can take the following forms:

- A *deterministic, memoryless policy* is a function $\pi : S \to A$, where $\pi(s)$ returns the action to execute in state $s \in S$ (see Section 2.2).
- A *stochastic, memoryless policy* is a function $\pi : S \to \text{Dist}(A)$, where $\pi(s)$ returns a distribution over the action to execute in state $s \in S$.
- A *deterministic, history-dependent policy* is a function $\pi : H \to A$, where H is the set of execution histories. An execution history $h \in H$ up to decision step k is the sequence of states visited and actions executed up to step $k-1$, and the state for step k. Formally, $h = s^1a^1...s^{k-1}a^{k-1}s^k$, where $s^l \in S$ and $a^l \in A$ are the state and action at decision step l respectively. For history $h \in H$, $\pi(h)$ returns the action to be executed given h.
- A *stochastic, history-dependent policy* is a function $\pi : H \to \text{Dist}(A)$, where $\pi(h)$ returns a distribution over the action to execute given history h.

CAMAS supports each of the policy types defined above. However, CAMAS requires that the policy π_i for robot r_i must return actions in A_i, i.e actions that can be executed by r_i. Further, π_i must be defined over the MRMA state space $S_{\mathcal{A}_J}$, the state space of r_i's single-robot model $S_{\mathcal{A}_i}$, or histories over either of these state spaces and $A_{\mathcal{A}_J}$ or A_i. This ensures the policies can be executed on an MRMA. Policies defined over $S_{\mathcal{A}_J}$ allow each robot to fully observe the joint MRMA state. MRMA states model action progress, allowing robots to make decisions based on the progress of others. While robot r_i is executing an action, π_i returns a dummy value $\oslash$, i.e. we assume r_i cannot change action in the middle of execution. With this, $\oslash$ is used in execution histories for states without immediate transitions, i.e. where all robots are executing actions. For partially observable models such as Dec-POMDPs [134], policies are often defined over observation-action histories, as the underlying state is unobservable. CAMAS can support these policies, but requires a function from MRMA states to observations to do so.

Algorithm 5.1 The CAMAS Algorithm.

Input: Policies π_i, $\forall i \in [\![1, n]\!]$, initial state $\bar{s} = (\bar{s}_{l,i}, \perp, ..., \bar{s}_{l,n}, \perp, \bar{s}_g)$, property φ
Output: Timed execution trace σ

$s \leftarrow \bar{s}$
$t \leftarrow 0$
$\sigma \leftarrow (s, t)$
while `terminationCondition`(φ) not met **do**
    // If there is at least one robot waiting to execute an action
    if $\exists i.\, s_{p,i} = \perp$ and $A_i(s) \neq \emptyset$ **then**
        // Choose the next robot to begin action execution
        $r_i \leftarrow$ `chooseRobot`(s)
        $a \leftarrow$ `policyAction`(π_i, σ)
        $d \leftarrow 0$
        // Sample successor using immediate transition function
        $s' \leftarrow$ sample from $\{s'' : \delta_{\mathcal{A}_J}(s, a, s'') \mid s'' \in S_{\mathcal{A}_J}\}$
    else
        $a \leftarrow \oslash$
        // Sample transition duration using exit rate
        $d \leftarrow$ sample from exponential distribution with rate $E(s)$
        // Sample successor using branching distribution
        $s' \leftarrow$ sample from $\{s'' : \frac{\Delta_{\mathcal{A}_J}(s,s'')}{E(s)} \mid s'' \in S_{\mathcal{A}_J}\}$
    end if
    $s \leftarrow s'$
    $t \leftarrow t + d$
    // Update the trace
    `append`(σ, a)
    `append`($\sigma, (s, t)$)
end while

5.3.2 The CAMAS Algorithm

In Alg. 5.1, we describe how to simulate multi-robot behaviour using CAMAS. Given a set of robot policies, the initial MRMA state, and a property we wish to evaluate, CAMAS returns a timed trace through the MRMA which describes simulated robot behaviour. A trace σ is a state-action sequence where states are annotated with the wall clock time they were reached. Formally, a trace up to decision step k is denoted $\sigma = (s^1, t^1)a^1...(s^{k-1}, t^{k-1})a^{k-1}(s^k, t^k)$, where s^l and a^l are the state and action for step l respectively, and t^l is the wall clock time s^l was reached.

Alg. 5.1 simulates multi-robot behaviour by sampling through the MRMA from the initial state at time 0. In the MRMA, immediate transitions representing robot

action choice fire before exponential transitions, as they occur instantaneously. At each state, CAMAS first checks for immediate transitions, i.e. robots who need to choose an action (line 6). If there are robots awaiting action choice, we choose one arbitrarily (line 8), and select an action using their policy (line 9). In most states there is at most one robot awaiting action choice, as no two actions terminate at the same instant in continuous time. However, in the initial state, or states where robots must synchronise, e.g. for collaborative handovers, multiple robots need to choose an action. Only one transition fires in each MRMA state, and so in lines 6-12 we select robot actions sequentially in successive MRMA states. As immediate transitions are instantaneous, each robot chooses their action at the same instant in time. The `policyAction` function in line 12 extracts the correct policy input from the current trace, i.e. the current state for memoryless policies, and the execution history for history-dependent policies. We then sample a successor state from the immediate transition function $\delta_{\mathcal{A}_J}$ (line 12). If all robots are executing an action, i.e. there are no immediate transitions, we resolve the race condition induced by the exponential transitions by sampling a duration using the exit rate (line 16), and a successor state using the branching distribution (line 18). Upon sampling the successor, we append the current action to the trace, which is $\oslash$ if an exponential transition was fired (line 23). We also append the successor state to the trace alongside the updated wall clock time computed by adding the transition duration, which is zero for immediate transitions (line 24).

CAMAS samples through the MRMA until a problem-dependent termination condition is met (line 4). For example, we may terminate each simulation after a fixed duration, or when a certain condition over states is satisfied. For unbounded properties, such as invariant safety properties, the termination condition may require the existence of an absorbing state, or a bottom strongly connected component to have been reached in the MRMA [177].

To evaluate multi-robot policies and address **Q4 - Policy Evaluation**, we analyse a set of traces obtained with Alg. 5.1. For example, to evaluate a task's expected completion time, we find the wall clock time the task is completed in

each trace, and compute the mean. Alternatively, to evaluate the probability of satisfying a temporal logic specification, we run each trace through an automaton that represents the specification, and count the number of satisfying traces [164].

Example 5.4. Consider Fig. 5.2, which demonstrates a CAMAS simulation run through MRMA $\mathcal{A}_J$ starting at time t. Here, each robot's local state is its current node, $S_{l,i} = V$, and robot policies are deterministic, memoryless, and defined over the joint MRMA state, i.e. $\forall r_i,\ \pi_i : S_{\mathcal{A}_J} \to A_i$. In state $(v_1, \perp, v_3, \mu)$ in Fig. 5.2b, robot r_1 chooses action $\pi_1(v_1, \perp, v_3, \mu) = (v_1, v_2)$. Assume that the context for (v_1, v_2) is c^2; the corresponding PTD is shown along the edge of (v_1, v_2) in Fig. 5.2a. The immediate transition samples the initial PTD state. From state (v_1, α, v_3, μ), r_1 wins two consecutive race conditions which progresses it along its PTD until it arrives at the new local state v_2, where the duration of these transitions are d and d' respectively. The corresponding trace is $((v_1, \perp, v_3, \mu), t)(v_1, v_2)((v_1, \alpha, v_3, \mu), t) \oslash ((v_1, \gamma, v_3, \mu), t + d) \oslash ((v_2, \perp, v_3, \mu), t + d + d')$.

5.4 Complexity Analysis

In this section, we outline the space complexity of the MRMA and the time complexity of CAMAS with respect to the number of robots n.

5.4.1 MRMA Space Complexity

First, the size of the joint system state space S_J is given by $\mathcal{O}((\max_{i \in [\![1,n]\!]} |S_{l,i}|)^n \cdot |S_g|)$, i.e. the product of each robot's local state space and the global state features, which is exponential in the number of robots. Next, we consider the PTD states. Let $\mathcal{P}_{max}$ be the largest PTD across ρ_E and ρ_N. For each enabled action in a state, we need at most $|S_{\mathcal{P}_{max}}|$ states for that action. In any MRMA state, there are at most $|A_{max}| \cdot n$ enabled actions, i.e. $|A_{max}|$ actions for each robot, where A_{max} is the largest robot action set. PTD evolution in the MRMA does not affect the local or global state features, and so from each $s \in S_J$ there are at most $\mathcal{O}(|S_{\mathcal{P}_{max}}| \cdot |A_{max}| \cdot n)$ states in the MRMA. Therefore, the MRMA size is

$\mathcal{O}((\max_{i\in[\![1,n]\!]} |S_{l,i}|)^n \cdot |S_g| \cdot |S_{\mathcal{P}_{max}}| \cdot |A_{max}| \cdot n)$, which remains exponential in the number of robots.

5.4.2 CAMAS Time Complexity

CAMAS employs lazy MRMA construction, only expanding MRMA states reached during simulation runs. We now consider the time complexity of a single CAMAS simulation run. Let a_{max} be the maximum number of actions executed by a robot in a single simulation. For each action, we visit a number of PTD states upper bounded by $|S_{\mathcal{P}_{max}}|$. Therefore, each robot visits at most $\mathcal{O}(a_{max} \cdot |S_{\mathcal{P}_{max}}|)$ states. The number of states visited per robot is unaffected by the team size, and so we visit at most $\mathcal{O}(n \cdot a_{max} \cdot |S_{\mathcal{P}_{max}}|)$ MRMA states per simulation. At each state, we sample a successor state from a branching distribution of size n, which has worst-case complexity $\mathcal{O}(n)$ [246]. Therefore, the time complexity of CAMAS is $\mathcal{O}(n^2 \cdot a_{max} \cdot |S_{\mathcal{P}_{max}}|)$, which is quadratic in the number of robots.

5.5 Experiments

In this section, we use CAMAS to validate the MRMA by predicting the behaviour of an MRS simulated in the Gazebo physics simulator [155]. If CAMAS closely matches what we observe in Gazebo, the MRMA is able to accurately capture high-level multi-robot behaviour. We do this for scalar and vector contexts, and compare against two baselines. Further, we compare how using random navigation and edge partitioning for data collection affect the MRMA duration models. Gazebo simulations are run on AWS RoboMaker. All other simulations are run on Ubuntu 18.04, with an Intel Core i9-10900K CPU@3.7GHz and 32GB of RAM. All software is written in Python.

5.5.1 Experimental Setup

Experimental Environments

In our experiments, we model the effects of congestion (see Chapter 4.2). We consider two environments: a warehouse (Fig. 5.3), and a supermarket (Fig. 5.4). The warehouse is small, forcing congestion. The larger supermarket allows us

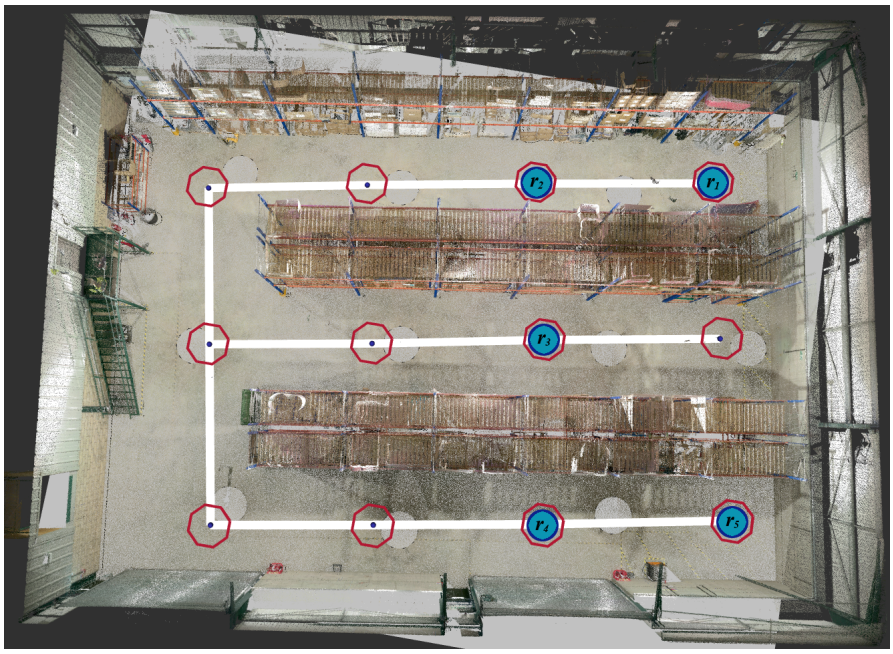


Figure 5.3: The warehouse environment with topological map overlaid. Blue circles show initial robot locations.

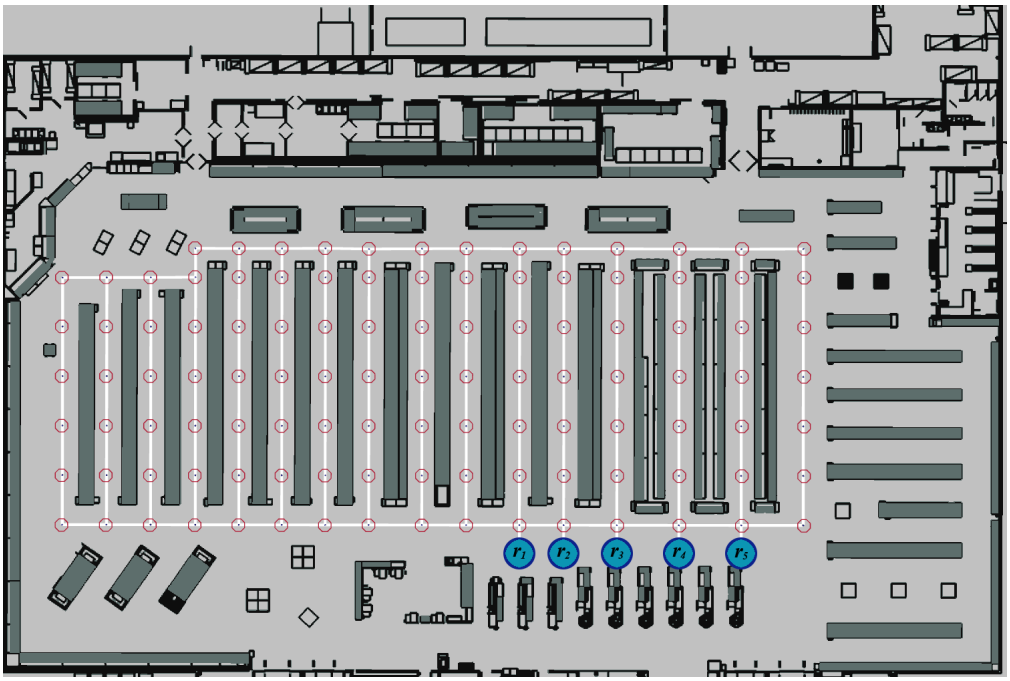
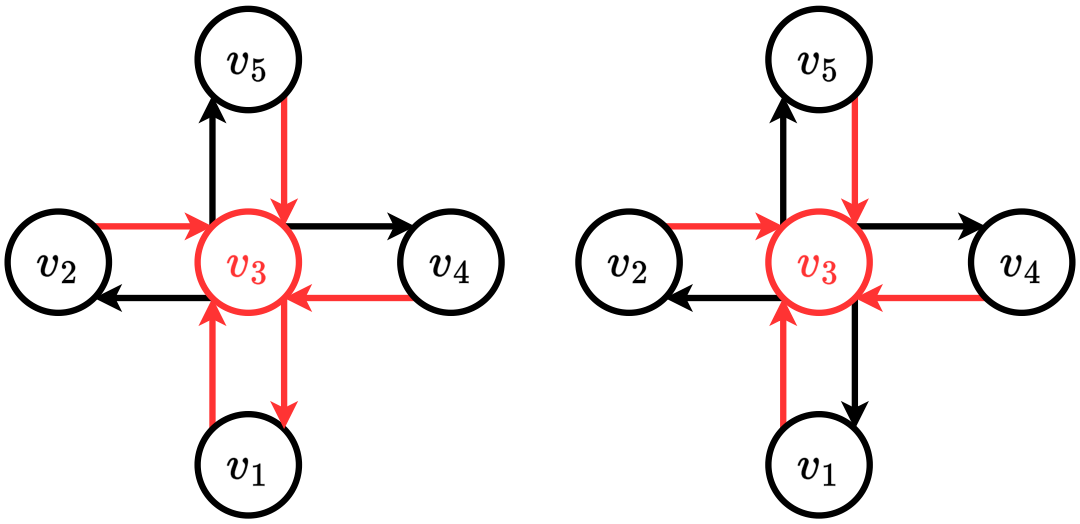


Figure 5.4: The supermarket environment with topological map overlaid. Blue circles show initial robot locations.



(a) The map resource group for edge (v_1, v_3). **(b)** The map resource group for node v_3.

Figure 5.5: Experimental map resource groups for Chapter 5. Nodes and edges highlighted in red are included in the map resource group.

to assess the MRMA and CAMAS over longer horizons. Both environments are rendered in Gazebo [155], and a topological map is created for each. We observe that robots travelling on different edges to the same node often cause temporary blockages, increasing navigation durations (see Example 4.1). Therefore, we define the map resource group at an edge $(v_1, v_2) \in E$ as $g((v_1, v_2)) = \{(v_1, v_2), (v_2, v_1), v_2\} \cup \{(v', v'') \in E \mid v'' = v_2\}$, i.e. the edge itself, the reverse edge, the target node v_2, and all edges that arrive at v_2 (see Fig. 5.5a). For a node $v \in V$, $g(v) = \{v\} \cup \{(v', v'') \in E \mid v'' = v\}$, i.e. all edges that arrive at v, and v itself (see Fig. 5.5b).

Data Collection

We use Gazebo to build navigation duration and failure datasets under each context by simulating five Clearpath Jackals navigating using Move Base Flex [247] for 24 hours of real time. We do this for random navigation and edge partitioning, and fit PTDs from the data using the method in [51] (see Section 4.4). For the experiments in this section, we do not use interaction bands, i.e. each context has a unique duration distribution (see Subsection 4.3.2). Robots may fail due to crashes or motion planner failure when navigating through congestion. From the observed failure probabilities, we add uncertain outcomes to immediate transitions in the

MRMA, and sample the outcome in CAMAS when robots choose to navigate. If a robot fails, it stops on the failed edge and executes no further actions, increasing congestion for any future robot navigating that edge.

During data collection, we may not observe all possible contexts. Further, we exclude data for contexts with less than 10 samples. For contexts without data, we use the PTDs and success probabilities of the nearest context for that action with sufficient data, which for scalar contexts we compute using the difference in the number of robots between contexts. The nearest vector context is computed by summing the robot differences for each map resource in the context.

Experimental Baselines

To validate the MRMA we compare CAMAS to Gazebo simulations identical to those used for data collection. We also compare against two baseline models which allow us to assess our use of contexts and stochastic action durations. The first baseline ignores contexts. For each action, we fit a single PTD and failure probability using all of the data collected for that action. Therefore, the distribution over contexts is fixed to that observed during data collection. The second baseline assumes deterministic action durations, but uses vector contexts. The duration for an action and context is the mean value observed during data collection. Runs can be simulated through these baseline models identically to CAMAS.

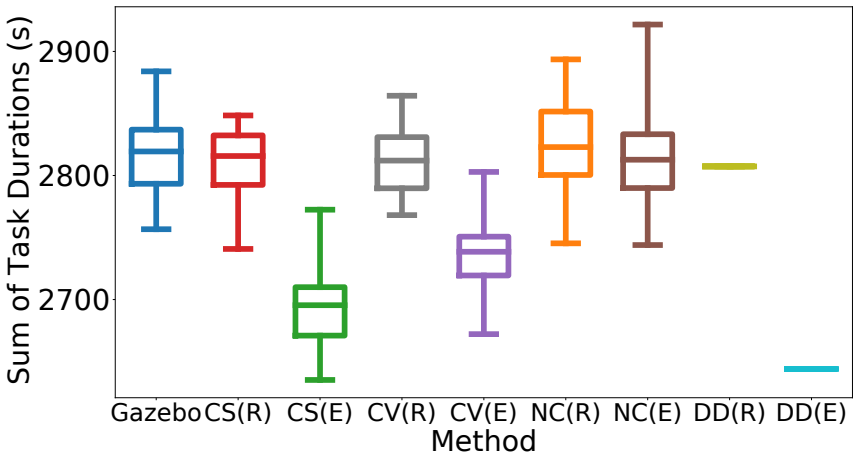
Experimental Problem

In both environments, each robot completes five navigation tasks, where each task requires the robot to visit two locations in sequence. This is similar to pick and place tasks, but without manipulation. We focus on navigation problems to assess how accurately our models capture congestion. The first location in each task is selected randomly, but kept consistent between all methods. The second location, which we refer to as the return location, is fixed for each robot across all of their tasks, as shown by the blue circles in Fig. 5.3 and Fig. 5.4. The first and second locations for a task are never equal. In the supermarket, the fixed return locations are set close to each other to force congestion. For the supermarket, we also consider

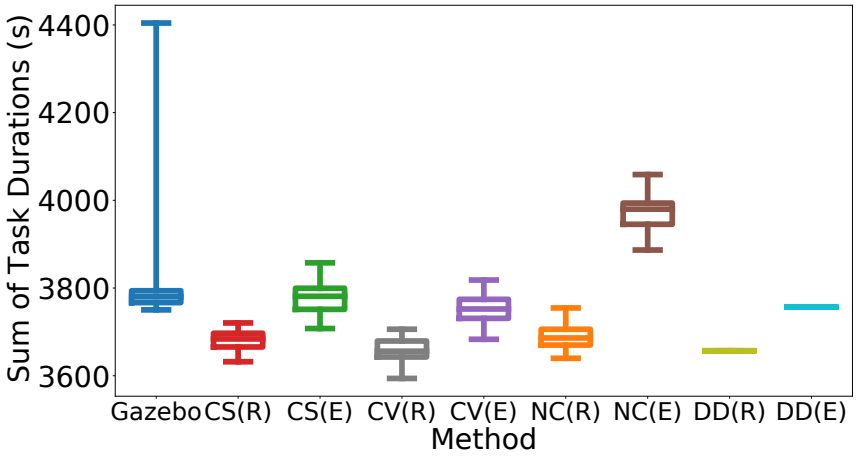
a variant where the return locations for each task are randomised to compare duration datasets when congestion is sparse during execution. For each method, each robot has three local state features: their location, their current task, and the location for that task they are currently navigating towards. There are no global state features. CAMAS and the baseline simulators terminate when all robots have finished their five tasks or stopped due to failure.

5.5.2 Evaluating Duration-Based Metrics

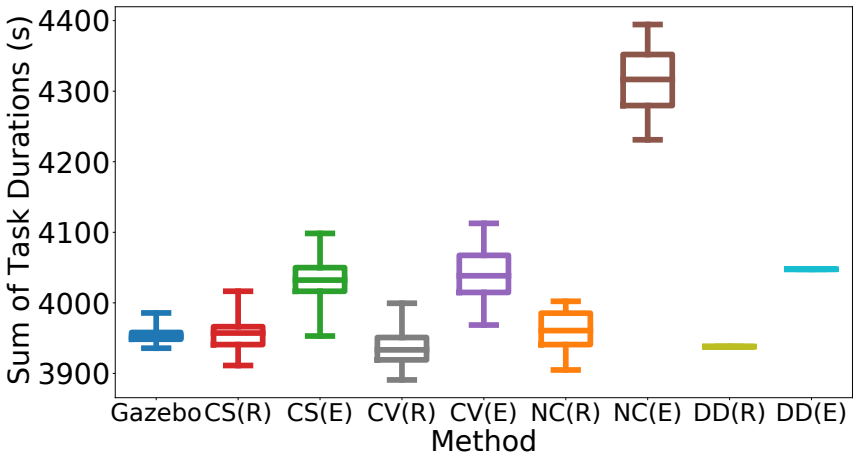
To validate that the MRMA accurately captures high-level multi-robot behaviour, we first analyse how closely CAMAS predicts task durations observed in Gazebo. To do this, we compare 40 successful Gazebo runs (i.e. no robots fail) to 40 runs of CAMAS and the baseline simulators, where all actions succeed with probability 1. In Fig. 5.6, we measure the sum of task durations as a qualitative comparison between each method and Gazebo. Further, we compare the empirical duration distributions for each task, which provides a more granular insight into the performance of each method. For each method and task, we compute the error against Gazebo using a two-sided Kolmogorov-Smirnov (KS) test, which computes the maximum difference between empirical cumulative distribution functions (CDFs) [248]. A low KS result means the distributions are similar, i.e. the method closely matches the task duration distribution observed in Gazebo. For each method/problem setup, we sum the KS results for each task, and present the summed results in Table 5.1. Moreover, in Fig. 5.7 we show the empirical CDFs for each method across two tasks in the warehouse. To provide representative results, we select the two tasks where the sum of each method's KS result is lowest and highest. In all plots and tables, we write CS and CV for CAMAS with scalar and vector contexts respectively, NC for the no context baseline, and DD for the deterministic duration baseline. We follow these with (R) or (E) to denote whether random navigation or edge partitioning was used for data collection.



(a) Warehouse.



(b) Supermarket (fixed return).



(c) Supermarket (randomised return).

Figure 5.6: The sum of task durations for each method.

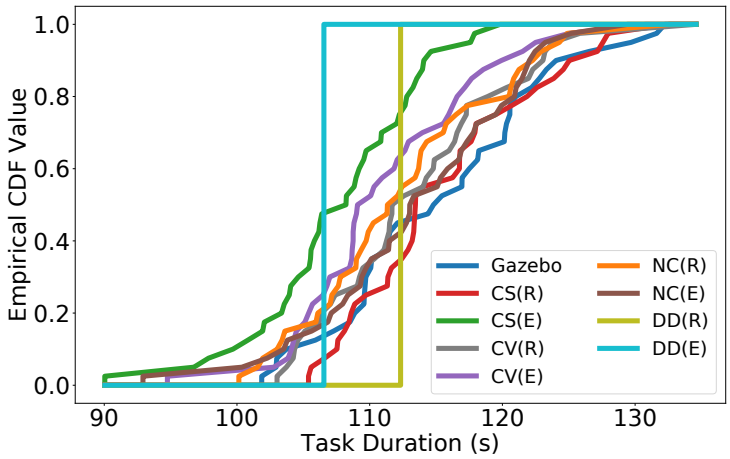
Table 5.1: The error between Gazebo and each method, where each value is the sum over the KS test results for each task. Bold values give the lowest error for that problem setup. All values are unitless.

Method	Warehouse	Supermarket (fixed return)	Supermarket (randomised return)
CS(R)	8.55	15.57	**13.45**
CS(E)	12.27	**11.80**	15.05
CV(R)	**8.00**	16.60	13.88
CV(E)	10.27	13.15	15.05
NC(R)	10.60	15.90	14.63
NC(E)	10.75	16.28	21.87
DD(R)	18.30	22.85	22.53
DD(E)	22.68	21.90	24.25

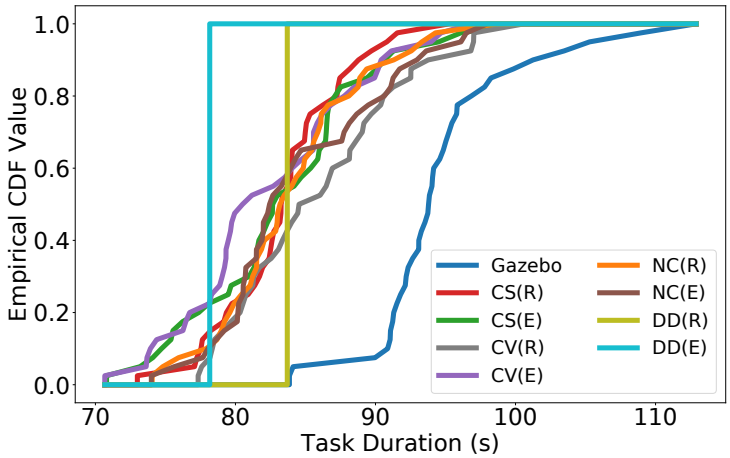
Results

A CAMAS variant gives the lowest error for each problem in Table 5.1, demonstrating the accuracy of the MRMA. The Gazebo simulations are inherently uncertain, due to environmental factors and uncertainty caused by robot interactions (cf. Fig. 5.6). Because of this, the deterministic duration baseline gives the highest error, as though it observes action contexts, it ignores duration uncertainty. The no context baseline performs well when the distribution over contexts observed during data collection matches what we observe during execution. However, this doesn't generalise, as highlighted when using edge partitioning, where robots frequently travel through the target edge. This causes heavy interactions during data collection which do not occur during the navigation tasks. Therefore, the no context baseline inaccurately models the frequency of interactions during execution, and so fails to capture the behaviour observed in Gazebo. The MRMA avoids this by using a distribution per context. This increases the data required for accurate duration models, but is a one time cost, as the duration distributions can be used for any problem in that environment.

CAMAS with vector contexts gives the lowest error in the warehouse. Vector contexts describe the spatiotemporal situation an action is executed in more precisely, which allows us to build distributions that better reflect the subtleties in different interaction scenarios. Further, in small environments we can collect sufficient data for most contexts. However, CAMAS with scalar contexts produces lower errors in the



(a) The empirical CDFs for the task where the sum of each method's KS result was lowest.

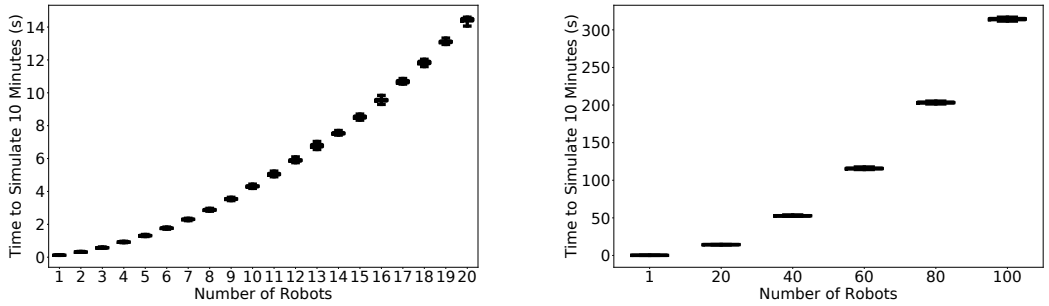


(b) The empirical CDFs for the task where the sum of each method's KS result was highest.

Figure 5.7: The empirical CDFs for each method across two tasks in the warehouse.

supermarket problems. In large environments, we are unlikely to gain sufficient data for many vector contexts, decreasing the accuracy of our duration models. Scalar contexts improve the duration models by aggregating data, thus building larger datasets.

Random navigation for data collection produces better duration models in the warehouse. As the environment is small, we obtain large samples for each edge and context. Further, random navigation enables contexts to occur in a natural way that is representative of real execution. Edge partitioning creates artificial scenarios



(a) Results from 1 to 20 robots. **(b)** Results from 1 to 100 robots.

Figure 5.8: CAMAS scalability results.

to observe high-interaction contexts, which creates less realistic datasets. However, edge partitioning improves performance for the supermarket with fixed return locations, where the return locations force congestion. The environment size prevents random navigation from gaining sufficient data for many observable contexts. Though edge partitioning collects data from less realistic scenarios, it provides a better view of high-interaction contexts, which improves model accuracy. Random navigation produces better duration models in the supermarket with randomised return locations as the data is more representative and there are few interactions, which renders much of the data collected by edge partitioning unnecessary.

5.5.3 CAMAS Simulation Speed

To assess the scalability of CAMAS, we simulate 40 runs of 10 minutes for team sizes ranging from 1 to 100 robots. Robots operate in the supermarket environment, and navigate randomly during the simulation. Fig. 5.8a shows scalability results for 1 to 20 robots in single robot increments, and Fig. 5.8b shows results for 1 to 100 robots in increments of 20. These results demonstrate the quadratic complexity of CAMAS (see Section 5.4). However, for 100 robots, CAMAS still runs 1.9 times faster than real-time on average, validating it as a scalable DES.

5.5.4 Evaluating Failure-Based Metrics

Next, we validate the MRMA accuracy on problems with uncertain outcomes by comparing CAMAS and the baselines to Gazebo on metrics concerning navigation

Table 5.2: Evaluation of the average tasks completed metric. In the first table, bold values have the closest mean to Gazebo, and highlighted cells show methods whose distribution is not statistically significantly different to Gazebo. In the second table, highlighted cells show methods whose distribution is not statistically significantly different to the method with the closest mean to Gazebo (written beneath the environment). Statistical significance is determined using a two-sided Mann-Whitney U test, where $p = 0.05$.

Average number of tasks completed, ± standard deviation.			
Method	**Warehouse**	**Supermarket (fixed return)**	**Supermarket (randomised return)**
Gazebo	22.07 ± 3.52	23.72 ± 2.79	24.13 ± 1.72
CS(R)	19.93 ± 3.83	16.45 ± 4.49	23.52 ± 2.41
CS(E)	19.68 ± 4.17	17.82 ± 3.21	17.40 ± 3.79
CV(R)	**20.70 ± 3.61**	**22.85 ± 2.14**	**24.17 ± 1.54**
CV(E)	20.08 ± 4.14	10.38 ± 3.13	15.70 ± 4.83
NC(R)	20.57 ± 3.46	20.77 ± 3.32	23.75 ± 1.91
NC(E)	19.62 ± 4.23	15.03 ± 4.33	15.45 ± 4.95
***p*-values against the method closest to Gazebo.**			
Method	**Warehouse CV(R)**	**Supermarket (fixed return) CV(R)**	**Supermarket (randomised return) CV(R)**
CS(R)	3.30×10^{-1}	1.21×10^{-12}	1.78×10^{-1}
CS(E)	1.97×10^{-1}	3.35×10^{-14}	2.11×10^{-18}
CV(R)	1.00	1.00	1.00
CV(E)	6.45×10^{-1}	2.34×10^{-21}	1.30×10^{-18}
NC(R)	8.80×10^{-1}	5.66×10^{-4}	1.82×10^{-1}
NC(E)	2.01×10^{-1}	6.10×10^{-18}	3.75×10^{-19}

failure. All methods use the failure probabilities obtained during data collection. We carry out 60 simulations of Gazebo, CAMAS and the baseline simulators, where navigation may fail, and consider three metrics for comparison:

1. *The average number of tasks completed.* A task is completed if a robot does not fail during its execution.

2. *The average number of robots succeeded.* A robot succeeds if it completes all five of its tasks.

3. *The probability of a run being successful.* A run is successful if all five robots complete all of their tasks without failure.

Table 5.3: Evaluation of the successful robots metric, following the same convention as Table 5.2.

Average number of robots succeeded, ± standard deviation.			
Method	**Warehouse**	**Supermarket (fixed return)**	**Supermarket (randomised return)**
Gazebo	3.73 ± 0.98	4.40 ± 1.17	4.63 ± 0.71
CS(R)	3.53 ± 1.06	1.92 ± 1.42	4.40 ± 0.88
CS(E)	3.45 ± 1.13	2.75 ± 0.87	2.67 ± 1.07
CV(R)	3.65 ± 1.01	$\mathbf{4.22 \pm 0.71}$	$\mathbf{4.68 \pm 0.53}$
CV(E)	3.57 ± 1.09	0.63 ± 0.68	2.4 ± 1.16
NC(R)	$\mathbf{3.75 \pm 0.83}$	3.40 ± 1.11	4.53 ± 0.64
NC(E)	3.37 ± 1.17	1.98 ± 1.18	2.23 ± 1.19
***p*-values against the method closest to Gazebo.**			
Method	**Warehouse NC(R)**	**Supermarket (fixed return) CV(R)**	**Supermarket (randomised return) CV(R)**
CS(R)	2.91×10^{-1}	2.93×10^{-14}	1.22×10^{-1}
CS(E)	1.60×10^{-1}	7.08×10^{-14}	4.59×10^{-18}
CV(R)	6.28×10^{-1}	1.00	1.00
CV(E)	4.22×10^{-1}	4.98×10^{-22}	1.46×10^{-18}
NC(R)	1.00	4.51×10^{-5}	1.79×10^{-1}
NC(E)	9.06×10^{-2}	2.85×10^{-17}	3.28×10^{-19}

We provide results for these metrics in Tables 5.2, 5.3, and 5.4 respectively. We omit results for the deterministic duration baseline as failures are modelled identically to the MRMA with vector contexts.

Results

CAMAS with vector contexts under the random navigation dataset produces the closest means to Gazebo, except for the average robots succeeded in the warehouse. In this case, the no context baseline is closest to Gazebo, as the observed distribution over contexts during data collection is close to that observed during the navigation tasks. Note that the successful run probability results in Table 5.4 are typically furthest from Gazebo. This metric is highly sensitive, as if one task fails, the whole run is considered a failure. Therefore, a small change in a task's failure probability can significantly affect the successful run probability.

Edge partitioning leads to poorer predictions of navigation failure, as the failure

Table 5.4: Evaluation of the successful run metric, following the same convention as Table 5.2.

The probability of a run being successful.			
Method	**Warehouse**	**Supermarket (fixed return)**	**Supermarket (randomised return)**
Gazebo	0.283	0.717	0.767
CS(R)	0.183	0.067	0.617
CS(E)	0.200	0.017	0.033
CV(R)	**0.233**	**0.383**	**0.717**
CV(E)	0.217	0.000	0.033
NC(R)	0.167	0.200	0.600
NC(E)	0.150	0.017	0.033
***p*-values against the method closest to Gazebo.**			
Method	**Warehouse CV(R)**	**Supermarket (fixed return) CV(R)**	**Supermarket (randomised return) CV(R)**
CS(R)	5.02×10^{-1}	3.53×10^{-5}	2.47×10^{-1}
CS(E)	6.59×10^{-1}	5.74×10^{-7}	1.37×10^{-14}
CV(R)	1.00	1.00	1.00
CV(E)	8.28×10^{-1}	1.08×10^{-7}	1.37×10^{-14}
NC(R)	3.63×10^{-1}	2.78×10^{-2}	1.80×10^{-1}
NC(E)	2.48×10^{-1}	5.74×10^{-7}	1.37×10^{-14}

probabilities of the target edges are shared amongst all equal length edges, which may not hold in practice. Even if this probability is small, if robots execute long routes the probability of sampling a failure quickly increases. This is apparent in the supermarket with randomised return locations, where all edge partitioning methods produce distributions statistically different from Gazebo.

Vector contexts identify different interaction scenarios more precisely than scalar contexts, which aggregate multiple vector contexts, and combine their failure probabilities. Though data aggregation in the scalar context will decrease this probability, it results in less accurate models of navigation failure. Therefore, for the same dataset, vector context means are often closer to Gazebo than for scalar contexts.

Comparing the empirical distributions for the metrics in Tables 5.2-5.4, CAMAS is not always statistically similar to Gazebo, as modelling assumptions in the MRMA prevent us from perfectly capturing multi-robot behaviour. However, the MRMA

still allows us to closely predict the means in Tables 5.2-5.4, while providing faster simulations. For example, in the supermarket with fixed return locations, no method is statistically similar to vector CAMAS with random navigation, which produces the closest means, but is not statistically similar to Gazebo. In the warehouse, all methods perform statistically similarly, as the environment is small enough to collect sufficient datasets, and the contexts observed during data collection match execution. Further, in the supermarket with randomised return locations, all random navigation methods are statistically similar to Gazebo, as there is little congestion during random data collection and execution.

5.6 Discussion

The MRMA captures high-level multi-robot behaviour more accurately than models which do not use contexts or stochastic duration models. The MRMA is useful for modelling problems where different contexts show distinct duration profiles, such as the multi-robot navigation problems addressed in this thesis. If there is little difference in duration effects between contexts, the MRMA will perform similarly to the no context baseline. The MRMA observes the context at the instant a robot begins action execution and selects the PTD for that context. This assumes that the context is constant for the duration of the action, which often doesn't hold during execution, leading to modelling inaccuracies in the MRMA. For example, a new robot may join an edge in the middle of another's navigation action, causing congestion. However, for multi-robot navigation problems, the context is unlikely to change significantly during an action. Moreover, the effects of these changes will be captured implicitly in our duration distributions, assuming they also occur during data collection. If the context changes rapidly during execution, such as for autonomous vehicles navigating through high speed traffic, the MRMA may not be an appropriate modelling formalism, as identifying the context at the start of an action gives limited information on the action duration. In this instance, each duration distribution will capture the effects of many interaction scenarios, similar to the no context baseline, which fails to capture execution-time robot

behaviour, as the distribution over contexts is fixed to that observed during data collection. This could be mitigated by adding intermediate nodes to the topological map which split up navigation actions, but this would increase the MRMA size. Alternatively, the range of the map resource groups could be increased, however this would increase the number of vector contexts.

Vector contexts allow us to discriminate between different interaction scenarios more precisely than scalar contexts, but the increased number of contexts per action makes data collection more difficult. If limited data is available, scalar contexts may provide better task duration estimates. For duration-based metrics, edge partitioning is beneficial in large environments with frequent robot interactions. However, problems arise when generalising uncertain action outcomes. For smaller environments, or those with fewer interactions, random navigation collects more realistic execution data, improving performance.

CAMAS rapidly and accurately evaluates task-level metrics of multi-robot behaviour by abstracting low-level robot dynamics into context-dependent duration distributions. As a result, CAMAS is unsuitable for evaluating lower level behaviour, such as the contact points of a legged robot. In that instance, more realistic physics-based simulators such as Gazebo [155] should be used for policy evaluation.

5.7 Summary

In this chapter, we have presented an MRMA formulation that captures asynchronous multi-robot execution in continuous time. The MRMA captures robot dynamics using the context-dependent duration distributions introduced in Chapter 4, where the context is observed at the instant a robot chooses an action. Further, we adapt scalar and vector contexts (see Subsection 4.3.1) to the MRMA, which describe the presence of other robots near a robot's action location, and allow us to identify robot interactions such as congestion. The MRMA is the first model to explicitly consider how robot interactions affect action durations. The MRMA has multiple applications in planning, reinforcement learning, model checking, and simulation. In this chapter we presented CAMAS, a scalable DES that samples through the MRMA

to accurately evaluate task-level metrics of multi-robot performance given a set of robot policies. CAMAS can simulate teams of 100 robots 1.9 times faster than real time, where this speed up increases for smaller MRSs. The MRMA and CAMAS address **Q2 - Modelling** and **Q4 - Policy Evaluation** respectively, and have been published in [25]. Empirically, the MRMA accurately captures the task-level behaviour of an MRS simulated in Gazebo [155], and is more accurate than models which ignore contexts and stochastic action durations. Though we focus on robotics, the MRMA and CAMAS are applicable to any multi-agent system.

6

Congestion-Aware Policy Synthesis for Multi-Robot Systems

Contents

6.1 Introduction 97
6.2 Time-Varying Markov Automata (TVMA) 99
6.3 Problem Formulation 100
6.4 Framework 101
6.5 Modelling the Evolution of Congestion 103
6.5.1 Representing Congestion 103
6.5.2 Computing Congestion Probabilities 104
6.5.3 Defining a PRT 106
6.6 Planning Under Congestion 106
6.6.1 Single-Robot Planning Models 107
6.6.2 LRTDP for TVMAs 110
6.6.3 Constructing a Robot's Route CTMC 114
6.6.4 Complexity Analysis 116
6.7 Policy Execution Model Refinement 117
6.8 Experiments 120
6.8.1 Experimental Setup 120
6.8.2 Validation in Synthetic Environments 122
6.8.3 Validation in a Simulated Environment 132
6.9 Discussion 136
6.10 Summary 137

6.1 Introduction

To address **Q3 - Planning** and synthesise efficient robot behaviour, we must explicitly reason over temporal uncertainty and the effects of robot interactions such as congestion during planning. One planning approach is to use the MRMA (see Chapter 5), which accurately captures multi-robot execution under stochastic durations and robot interactions. However, the MRMA scales exponentially in the number of robots (see Section 5.4). Empirically, this prevents us from synthesising behaviour for more than small two robot problems using optimal planning methods. An alternative approach is to constrain the problem to prevent interactions from occurring. MAPF solvers achieve this by synthesising plans that prevent robots from being in the same location simultaneously [18]. However, this forces robots to take conservative routes to avoid each other, resulting in inefficient behaviour. To mitigate the poor scalability of planning on the MRMA while synthesising efficient robot behaviour, in this chapter we present a novel framework for probabilistic planning that decouples the robots while explicitly modelling congestion. We address multi-robot navigation tasks on a context-aware topological map, and use context-dependent PTDs to model robot navigation under different congestion scenarios. Our framework reasons over the effects of congestion on navigation duration, allowing a robot to choose a longer route when shorter alternatives are likely to present heavy congestion; or select a slightly congested route that is likely to be quicker than longer, less congested alternatives.

Given a policy describing a robot's navigation route, we use the corresponding PTDs to construct a CTMC that captures the continuous-time behaviour induced by the policy. As well as contributing to **Q2 - Modelling**, we use these CTMCs to compute the probabilities of encountering different congestion levels at different times across the map. The PTDs, CTMCs and congestion information are managed in a structure called a *probabilistic reservation table* (PRT), which we use in a sequential planning framework, where each robot plans using knowledge of those who planned before. This framework synthesises suboptimal behaviour, as robots ignore those who are yet to plan, but avoids the complexity of planning on the MRMA. We

use the PRT to construct single-robot *time-varying Markov automata* (TVMA), a novel extension of MA for continuous-time planning problems under time-varying dynamics which contributes towards **Q2 - Modelling**. Crucially, a TVMA allows the planning robot to capture the influence of the other robots by encoding their presence as *congestion probabilities*. Generally, congestion distributions capture uncertainty over the context, which cannot be observed directly as in an MRMA. The single-robot TVMAs decouple the robots in the MRMA, where from the perspective of a single robot, the presence of other robots introduces non-stationarity (see Subsection 3.1.7). To synthesise robot behaviour, we present an approximate TVMA solution method which extends LRTDP [87] to incrementally build and solve an MDP abstraction of the TVMA. LRTDP synthesises policies quickly by avoiding unnecessary portions of the state space (see Subsection 3.1.2). The policies we synthesise are robust to unexpected delays since a robot's arrival time to a node is included in the policy's state space. Therefore, if a robot arrives late to a location, its policy action may change. The congestion-aware planning framework was first introduced in [23], and later expanded in [24].

As well as policy synthesis, we also want to evaluate the policies we synthesise prior to execution, as specified in **Q4 - Policy Evaluation**. For example, we may want to know the probability that a robot will arrive to its goal within a time bound. To address **Q4 - Policy Evaluation**, we require models which predict robot policy execution under congestion. We present an iterative procedure that refines single-robot CTMCs stored in the PRT to more accurately capture the effects of the other robots. We then evaluate the robot policies using model checking techniques to obtain probabilistic guarantees on the refined CTMCs. This is orthogonal to CAMAS in Chapter 5. CAMAS uses the MRMA, which captures joint multi-robot behaviour. Exact model checking techniques cannot be applied to the MRMA due to its size, and so CAMAS samples through the MRMA for policy evaluation. Conversely, the iterative approach in this chapter applies exact model checking techniques to approximate single-robot models in a multi-robot setting. This presents a trade off between applying approximate techniques on

accurate models, and applying exact techniques on approximate models. The iterative refinement procedure was first introduced in [24]. The contributions in this chapter are validated with experiments on synthetic data and in a robotic simulator, where we compare our framework against two baselines which represent either end of a spectrum of techniques for handling multi-robot interactions during execution.

6.2 Time-Varying Markov Automata (TVMA)

In this chapter, we address SSPs over labelled TVMAs, denoted TVMA-SSPs. TVMAs extend MAs [56] to include immediate and exponential transitions with time-varying dynamics [128, 129]. In Section 6.6, we use TVMA-SSPs to model the single-robot planning problem under congestion.

Definition 6.1. A *labelled TVMA* is a tuple $\mathcal{A}^{\text{TV}} = \langle S, \bar{s}, A, \delta, \Delta, AP, Lab \rangle$, where S, $\bar{s}$, A, AP, and Lab are as in an MA. $\delta : S \times A \times S \times \mathbb{R}_{\geq 0} \to [0, 1]$ is a time-varying immediate transition function, where $\delta(s, a, s', t)$ returns the probability of instantaneously transitioning to state s' after action a was taken in state s at time t. $\Delta : S \times S \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{>0}$ is a time-varying exponential transition function, where $\Delta(s, s', t)$ returns the rate between states s and s', given that we arrived at state s at time t.

The semantics of TVMA transitions and the exit rate are similar to an MA, described in Section 2.5, but dependent on the arrival time to a state. For example, the probability an exponential transition between s and s' fires within time d, given we arrived at state s at time t, is given by $1 - e^{-\Delta(s,s',t)\cdot d}$. Similar to MAs, TVMAs are not required to have an associated cost function (see Section 2.5). To define a TVMA-SSP, we augment a TVMA with a time-varying cost function and a set of goal states.

Definition 6.2. A *TVMA-SSP* is an extension of an SSP [37], defined as a TVMA $\mathcal{A}^{\text{TV}} = \langle S, \bar{s}, A, \delta, \Delta, AP, Lab \rangle$, a time-varying cost structure $c : S \times A \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{>0}$ and a set of goal states $G \subseteq S$. The goal of a TVMA-SSP is to find a time-varying policy $\pi : S \times \mathbb{R}_{\geq 0} \to A$ that minimises the expected cost to reach a state in G.

For a time varying policy π, $\pi(s,t)$ represents the action the robot should take, given it arrived at state s at time t. TVMA-SSPs are subject to the same conditions as MDP SSPs [38] described in Section 2.2, i.e. there must be at least one proper policy, all improper policies must have infinite value, and all goal states must be absorbing.

6.3 Problem Formulation

In this section, we formulate the problems of planning (i.e. policy synthesis, Problem 6.1 below) and policy evaluation (Problem 6.2 below) under congestion, which correspond to **Q3 - Planning** and **Q4 - Policy Evaluation** respectively. For policy synthesis, we focus on multi-robot navigation tasks, where each robot must navigate to its goal location. However, the techniques we present can be generalised to other problems. First, we define local timed policies over a context-aware topological map, the space of policies we search over during policy synthesis.

Definition 6.3. A navigation policy for a robot r_i acting on a context-aware topological map $\mathcal{T}_C = \langle V, E, C, \rho_E \rangle$ is defined as $\pi_i : V \times \mathbb{R}_{\geq 0} \to E \cup \{wait\}$. $\pi_i(v,t)$ represents the action r_i should take when it arrives at location v at time t. If $\pi_i(v,t) = e \in E$, then edge e should be traversed. If $\pi_i(v,t) = wait$ then robot r_i should wait at node v.

Note that the definition above only assumes *local* information for robot r_i, namely its location and the current time. This allows the policy to be executed without communication with the other robots, making our solution resilient to communication failures. We now define the multi-robot policy synthesis problem we solve in this chapter.

Problem 6.1. Let $R = \{r_1, ..., r_n\}$ be a set of robots acting on a context-aware topological map $\mathcal{T}_C = \langle V, E, C, \rho_E \rangle$, where r_i has initial and goal locations $v_i^{init}, v_i^{goal} \in V$. Find $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$, where π_i is a policy for robot r_i, such that Π minimises the *makespan*, i.e. Π minimises the time until all robots have reached their goals.

Robots can navigate along edges of the topological map, and wait at nodes for congestion to decrease. We assume navigation actions always succeed and lead the robot to the desired location. We do this for simplicity of presentation, and because modelling stochastic outcomes increases the burden of data collection, requiring a PTD per outcome (see Chapter 5). However, this assumption holds for many navigation problems, as robots can often recover from motion planning failures due to minor collisions etc. to reach the destination. Under this assumption, uncertainty only arises from the temporal uncertainty over navigation duration. We also consider the problem of accurately evaluating properties of navigation policies.

Problem 6.2. Let π_i be a navigation policy for robot r_i, and φ_i a local property for r_i. Evaluate φ_i against the continuous-time behaviour induced by π_i under the effects of congestion.

In this chapter, we evaluate the expected time for a robot to arrive at its goal, and the probability a robot arrives at its goal within a time bound. However, our method extends to any property that can be model-checked against a CTMC, such as properties specified in continuous stochastic logic or linear temporal logic.

6.4 Framework

In this section we outline our approach to Problems 6.1 and 6.2 in Section 6.3. A high-level overview of the *congestion-aware planning framework* is given in Fig. 6.1. There is one global entity, the probabilistic reservation table (PRT). The PRT keeps track of robot policies, and represents robot behaviour as a set of *congestion distributions* over topological edges (see Subsection 6.5.2). Each robot plans to minimise the expected time to reach its goal on a single-robot TVMA which approximates the effects of the other robots. To consider the other robots, we assume robots plan sequentially, following a pre-defined priority order. For clarity of notation, a robot's priority is defined by its index, i.e. robot r_1 plans first, then r_2 etc. To minimise the makespan, we define the priority order using the heuristic in [107], where robots with longer routes have a higher priority. Robot r_i begins by

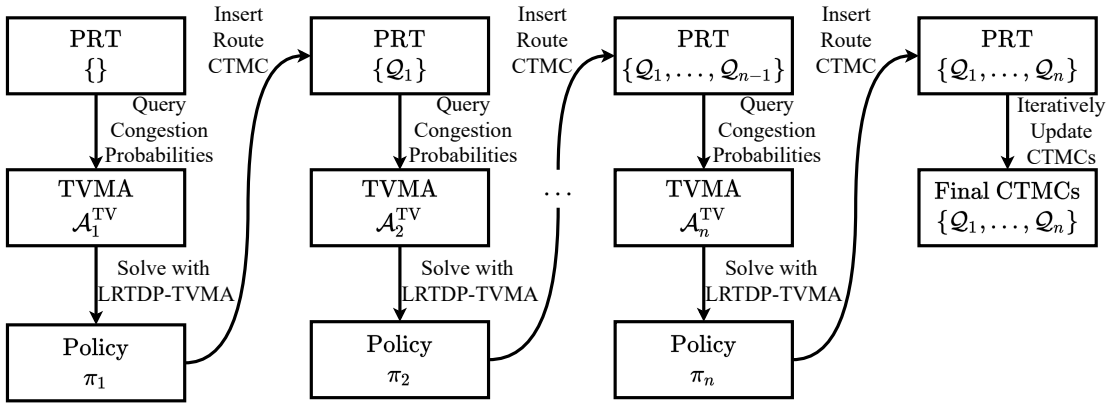


Figure 6.1: An overview of the congestion-aware planning framework. Robots plan sequentially. Each robot r_i queries the PRT to build its TVMA $\mathcal{A}_i^{\mathrm{TV}}$, which is solved to obtain a policy π_i. A route CTMC $\mathcal{Q}_i$ representing this policy is inserted into the PRT. After planning, all CTMCs can be updated iteratively to evaluate the behaviour induced by the synthesised policies, if required.

querying the PRT to obtain congestion probabilities summarising previous robot behaviour (see Subsection 6.5.2). These probabilities are used to construct a TVMA $\mathcal{A}_i^{\mathrm{TV}}$, from which we synthesise a policy π_i (see Subsection 6.6.1). From π_i, we construct a *route CTMC* $\mathcal{Q}_i$ and insert it into the PRT (see Subsection 6.6.3). The route CTMC captures the continuous-time behaviour induced by π_i, and is used to compute future congestion probabilities. Therefore, robot r_{i+1} uses the PRT to reason over the probabilistic effects of robots $r_1, ..., r_i$ in its TVMA. The PRT is initially empty. Therefore, the first robot who plans has no information about the other robots and so acts optimistically, as it believes it can follow the shortest path to the goal without interference. The final robot to plan takes into account all robots. Therefore, robots who plan earlier have less accurate planning models.

To accurately evaluate properties of the synthesised policies after planning (Problem 6.2), we require a model that accurately describes asynchronous multi-robot execution. One approach is to use the MRMA, and apply CAMAS to evaluate the behaviour induced by the synthesised policies. Alternatively, in this chapter we use the single-robot route CTMCs stored in the PRT to approximate the MRMA, which allows us to model check smaller, more tractable models. This introduces a trade off between approximate policy evaluation techniques on accurate models, and exact techniques on approximate models, which we discuss further in Section 6.7.

Prior to policy evaluation, we iteratively refine the route CTMCs to consider updated models of the other robots, particularly those that planned later (see Section 6.7). Refining a robot's CTMC does not modify the corresponding policy. The refined CTMC captures the continuous-time execution of the policy under the probabilistic effects of all other robot policies. This procedure is independent of the property being evaluated. Any single-robot property, such as expected task completion time or time-bounded reachability, can then be evaluated on the refined CTMCs using formal model checking techniques. The unrefined CTMCs can be evaluated in the same way, but are less accurate.

We proceed in Section 6.5 by presenting our probabilistic approach to modelling how congestion evolves during execution. In Section 6.6 we describe how to plan for each robot using congestion distributions, and in Section 6.7 we present the iterative CTMC refinement procedure.

6.5 Modelling the Evolution of Congestion

In this section, we present the probabilistic reservation table (PRT), which captures the stochastic evolution of congestion in continuous time.

6.5.1 Representing Congestion

In this chapter, we represent the effects of congestion on robot navigation using a context-aware topological map $\mathcal{T}_C = \langle V, E, C, \rho_E \rangle$, and use interaction bands over scalar contexts to describe congestion (see Subsections 4.3.2 and 4.3.1 respectively), where we pass interaction bands into ρ_E in place of the context. Scalar contexts use a single number of robots to capture robot presence on relevant map resources, and so each interaction band is a subset of $[\![0, n-1]\!]$. In Section 6.8, we analyse how the number of interaction bands affects planning performance.

In addition to moving, robots may wait at nodes until congestion is reduced. As navigation is stochastic, waiting is also treated stochastically using a waiting PTD $\mathcal{P}_\omega$, which is a two-state PTD with a single transition from the first to the second state with rate λ_ω. We then add a self loop, referred to as a *wait edge*,

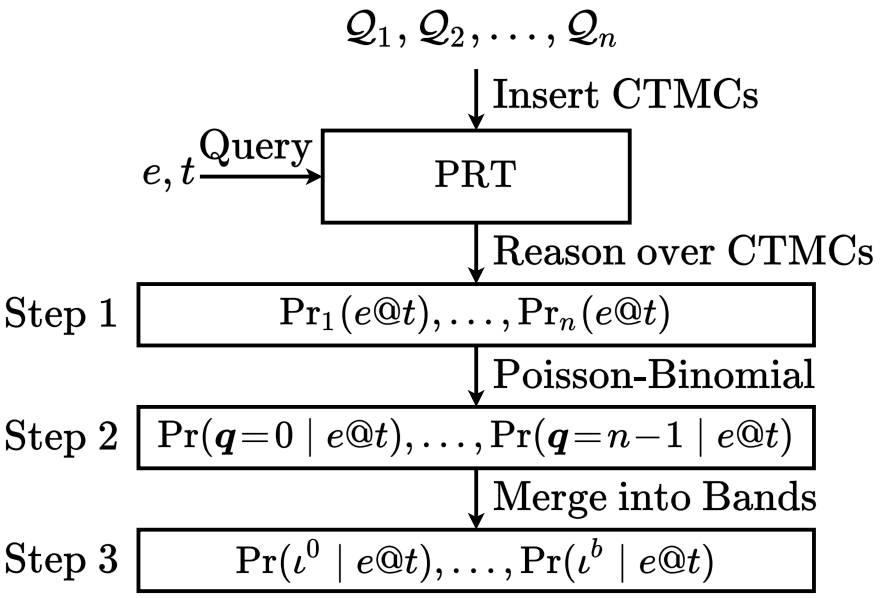


Figure 6.2: The pipeline for computing the congestion probabilities on edge e at time t using the PRT. Note, there may not be a CTMC for all robots at the time of use.

around each node in the context-aware topological map $\mathcal{T}_C$, i.e. $\forall v \in V,\ (v, v) \in E$. Wait edges have a single context and interaction band, as congestion can not be experienced when waiting, i.e. $\rho_E((v, v), \iota^0) = \mathcal{P}_\omega$, for all $v \in V$.

6.5.2 Computing Congestion Probabilities

The MRMA can deterministically identify congestion during planning. However, for scalability, in this chapter we plan on single-robot models. From a single robot's perspective, congestion evolves stochastically with time, as action durations are stochastic. When propagated over sequences of actions, this stochasticity prevents us from knowing where the other robots will be at a given time. To support planning, we require the *probability* of each interaction band occurring on edge e at time t, which we refer to as congestion probabilities, and compute as in Fig. 6.2.

To compute congestion probabilities, we require information about robot routes. We use a CTMC $\mathcal{Q}_i$ to represent the route of robot r_i. CTMCs naturally represent continuous-time robot behaviour, as we can concatenate context-dependent navigation PTDs to construct a route CTMC (see Subsection 6.6.3). Further, transient analysis of these CTMCs aids congestion probability computation [40], which we

discuss later in this section. First, we define the assumptions a route CTMC must satisfy, and in Subsection 6.6.3 we describe how to construct such a model.

Definition 6.4. Let $\mathcal{T}_C$ be a context-aware topological map. A *route CTMC* is a tuple $\mathcal{Q} = \langle S_\mathcal{Q}, init_\mathcal{Q}, \Delta_\mathcal{Q}, AP_\mathcal{Q}, Lab_\mathcal{Q} \rangle$, where $AP_\mathcal{Q} = E \cup \{goal\}$, i.e. the atomic propositions are comprised of the edges in $\mathcal{T}_C$ and a *goal* proposition. $Lab_\mathcal{Q}$ is defined such that all non-absorbing states in $\mathcal{Q}$ are labelled with a single topological edge, and absorbing states are labelled with *goal*, i.e. for all $s \in S_\mathcal{Q}$, $|Lab_\mathcal{Q}(s)| = 1$, and $Lab_\mathcal{Q}(s) = \{goal\}$ if and only if s is absorbing.

The route CTMC labelling function captures the robot's topological route. States labelled with edge $e \in E$ in route CTMC $\mathcal{Q}_i$ represent states where r_i is present on e. The route CTMC state space is the union of PTD state spaces, where each PTD represents an edge in r_i's route. Thus, each route CTMC state can be mapped to the edge modelled by the PTD it comes from. Absorbing route CTMC states represent route completion, and are labelled with proposition *goal*. The *goal* proposition is useful for policy evaluation, e.g. computing the expected time to reach the goal, as required by Problem 6.2. We discuss policy evaluation further in Section 6.7.

After a robot plans, it builds a route CTMC for its policy and stores it in the PRT. The first step to computing congestion probabilities in Fig. 6.2 is to consider the probability that each robot r_i is present on an edge in map resource group $g(e)$ at time t, denoted $\Pr_i(e@t)$. Recall that we consider all edges in the map resource group $g(e)$ since robots present on edges in $g(e)$ influence the congestion on e (see Subsection 4.2.1). To compute $\Pr_i(e@t)$, we use the *transient probabilities* of route CTMC $\mathcal{Q}_i$, as discussed in Subsection 2.3.1. We write $\Pr(s@t)$ to denote the probability of being in state $s \in S_{\mathcal{Q}_i}$ at time t. From the transient probabilities, we compute $\Pr_i(e@t)$ by summing across all states labelled with an edge in $g(e)$:

$$\Pr_i(e@t) = \sum_{\{s \in S_{\mathcal{Q}_i} \mid g(e) \cap Lab_{\mathcal{Q}_i}(s) \neq \emptyset\}} \Pr(s@t). \tag{6.1}$$

We compute the probability a robot is present on an edge using only its route CTMC. This assumes $\mathcal{Q}_i$ contains all necessary congestion information for

r_i. However, the true congestion probabilities change as new route CTMCs are added to the PRT. Under the sequential planning assumption, a robot's route is fixed at the point of insertion into the PRT. In Section 6.8, we demonstrate how this assumption still allows for effective reasoning over congestion. Moreover, in Section 6.7, we iteratively update the route CTMCs in the PRT to approximate the true congestion probabilities.

The second step in computing congestion probabilities for a new robot is to compute a discrete distribution over how many *other* robots are on an edge in $g(e)$ at time t, i.e. how congested the edge will be. We achieve this using the Poisson-binomial distribution, as described in Section 2.6, where the individual Bernoulli random variables model the presence of each robot, as distributed by $\Pr_i(e@t)$. We write $\Pr(q \mid e@t)$ to denote the probability of q other robots being present on an edge in $g(e)$ at time t. Finally, we group the probabilities for each number of robots into interaction bands. The probability $\Pr(\iota^j \mid e@t)$ of experiencing interaction band ι^j on edge e at time t, considering all edges that influence congestion on e, is given by:

$$\Pr(\iota^j \mid e@t) = \sum_{q \in \iota^j} \Pr(q \mid e@t). \tag{6.2}$$

6.5.3 Defining a PRT

A PRT is initialised with the set of robots R and context-aware topological map $\mathcal{T}_C$. A PRT is a table with an entry for each robot $r_i \in R$. The PRT entry for robot r_i, $\text{PRT}(r_i)$, stores its route CTMC $\mathcal{Q}_i$. Two functions are provided by the PRT: *update* and *cong*. The function $update(r_i, \mathcal{Q}_i)$ updates $\text{PRT}(r_i)$ by inserting route CTMC $\mathcal{Q}_i$. The function $cong(r_i, e, j, t)$ returns the congestion probability of r_i observing interaction band ι^j on edge e at time t, computed as in Fig. 6.2.

6.6 Planning Under Congestion

To solve Problem 6.1, we must synthesise a set of policies that guide each robot to its goal, and minimise the makespan. In Subsections 6.6.1 and 6.6.2, we outline a

scalable approach to planning using single-robot models, where congestion distributions summarise the team behaviour. Moreover, in Subsection 6.6.3 we discuss how to construct route CTMCs that describe continuous-time robot policy execution.

6.6.1 Single-Robot Planning Models

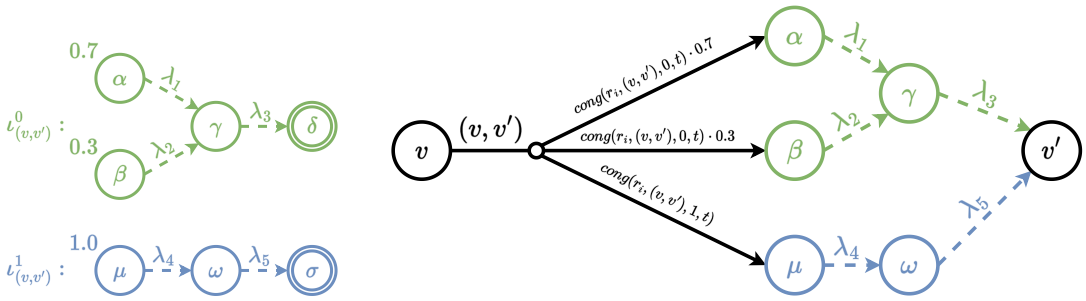
Congestion distributions vary with time, and action durations are represented as PTDs. Therefore, in Def. 6.5, we formulate robot r_i's planning model as a TVMA (see Section 6.2). Similar to Chapter 5, for PTD state $s \in S_{\mathbb{P}}$, let $\mathcal{P}(s)$, $e(s)$ and $b(s)$ denote the PTD, edge, and interaction band a state s belongs to respectively.

Definition 6.5. Given a PRT and context-aware topological map $\mathcal{T}_C$, the TVMA for robot r_i is a tuple $\mathcal{A}_i^{\mathrm{TV}} = \langle S_{\mathcal{A}_i^{\mathrm{TV}}}, \bar{s}_{\mathcal{A}_i^{\mathrm{TV}}}, A_{\mathcal{A}_i^{\mathrm{TV}}}, \delta_{\mathcal{A}_i^{\mathrm{TV}}}, \Delta_{\mathcal{A}_i^{\mathrm{TV}}}, AP_{\mathcal{A}_i^{\mathrm{TV}}}, Lab_{\mathcal{A}_i^{\mathrm{TV}}} \rangle$, where states are topological nodes or PTD states, $S_{\mathcal{A}_i^{\mathrm{TV}}} = V \cup S_{\mathbb{P}}$; the initial state is the robot's initial location, $\bar{s}_{\mathcal{A}_i^{\mathrm{TV}}} = v_i^{init}$; and $A_{\mathcal{A}_i^{\mathrm{TV}}} = E \cup \{wait\}$, i.e. robots can navigate on topological edges, or wait.

The immediate transition function $\delta_{\mathcal{A}_i^{\mathrm{TV}}}$ models robot action choice under congestion. From a topological node state s, we choose an outgoing edge e. For each interaction band ι^j on edge e there is a transition from s to the initial states s' of PTD $\rho_E(e, \iota^j)$. The transition probability is the congestion probability for ι_e^j multiplied with the initial state probability for s' in $\rho_E(e, \iota^j)$. Formally:

$$\delta_{\mathcal{A}_i^{\mathrm{TV}}}(s, e, s', t) = \begin{cases} cong(r_i, e, j, t) \cdot init_{\mathcal{P}(s')}(s') & \text{if } s \in V \text{ and} \\ & s = \mathtt{src}(e) \text{ and} \\ & \iota^j = b(s') \text{ and} \\ & s' \in S_{\rho_E(e,\iota^j)} \\ 0 & \text{otherwise.} \end{cases} \tag{6.3}$$

The exponential transition function $\Delta_{\mathcal{A}_i^{\mathrm{TV}}}$ captures the PTDs. For state s in PTD $\mathcal{P}$, the rate to a non-absorbing state $s' \in S_{\mathcal{P}}$ is defined according to $\Delta_{\mathcal{P}}$. Transitions to the absorbing state in $\mathcal{P}$ are rewired to the target of the edge the



(a) The PTDs for (v, v'). **(b)** The TVMA $\mathcal{A}_i^{\text{TV}}$.

Figure 6.3: An example TVMA for robot r_i at time t. Solid arrows are immediate transitions, dashed arrows are exponential transitions, and concentric circles represent PTD absorbing states. Note that in Fig. 6.3b, the immediate transition probabilities are dependent on t.

robot was traversing. Formally:

$$\Delta_{\mathcal{A}_i^{\text{TV}}}(s, s', t) = \begin{cases} \Delta_{\mathcal{P}(s)}(s, s') & \text{if } s, s' \in S_{\mathcal{P}(s)} \text{ and } s' \neq s^f_{\mathcal{P}(s)} \\ \Delta_{\mathcal{P}(s)}(s, s^f_{\mathcal{P}(s)}) & \text{if } s \in S_{\mathbb{P}} \text{ and } s' \in V \text{ and } s' = \texttt{trg}(e(s)) \\ 0 & \text{otherwise.} \end{cases} \tag{6.4}$$

The atomic propositions represent the action currently being executed, or the robot reaching its goal, i.e. $AP_{\mathcal{A}_i^{\text{TV}}} = E \cup \{wait, goal\}$. $Lab_{\mathcal{A}_i^{\text{TV}}}$ labels PTD states with the edge they model (or waiting), and the goal node with $\{goal\}$. Formally:

$$Lab_{\mathcal{A}_i^{\text{TV}}}(s) = \begin{cases} \{wait\} & \text{if } s \in S_{\mathcal{P}_\omega} \\ \{e(s)\} & \text{if } s \in S_{\mathbb{P}} \setminus S_{\mathcal{P}_\omega} \\ \{goal\} & \text{if } s \in V \text{ and } s = v_i^{goal} \\ \emptyset & \text{otherwise.} \end{cases} \tag{6.5}$$

In the TVMA, a robot arrives at a node and chooses an edge to navigate. There is a time-varying congestion distribution on this edge. For each band, there is a sequence of exponential transitions that describe the respective PTD. All PTDs finish at the location the robot arrives at. By constructing a set of single-robot TVMAs, we decouple the robots in the MRMA (see Chapter 5) for more scalable planning. We use TVMAs as the environment is non-stationary from the perspective of a single robot, i.e. the duration of robot actions changes with time,

as the robots do not know each other's locations (see Subsection 3.1.7). We capture this non-stationarity in the single-robot TVMAs using congestion distributions.

Example 6.1. Consider Fig. 6.3b, which shows a TVMA $\mathcal{A}_i^{\text{TV}}$ for robot r_i at time t. From the state at node v in $\mathcal{A}_i^{\text{TV}}$, robot r_i can choose edge (v, v'), which has two PTDs, one for each interaction band (see Fig. 6.3a). There are then time-varying immediate transitions for each interaction band and PTD initial state, where the transition probabilities combine the congestion probabilities and PTD initial state probabilities. For example, $\delta_{\mathcal{A}_i^{\text{TV}}}(v, (v, v'), \alpha, t) = \mathit{cong}(r_i, (v, v'), 0, t) \cdot 0.7$, which changes with t. After the immediate transitions, $\mathcal{A}_i^{\text{TV}}$ captures the PTDs for (v, v'), where upon finishing a PTD, we transition to the state at node v'.

As discussed in Section 2.5, policy synthesis algorithms for MA are computationally expensive, due to the combination of immediate and exponential transitions. To improve scalability, we pose the planning problem for r_i as a TVMA-SSP, and minimise the expected time to reach the goal, instead of solving for more complex objectives such as time-bounded reachability. As such, the TVMA-SSP cost function only considers expected action durations, which mitigates the complexity of digitisation-based methods [57].

Definition 6.6. The *TVMA-SSP* for robot r_i is defined as a TVMA $\mathcal{A}_i^{\text{TV}}$ according to Def. 6.5, a cost function $c_i : S_{\mathcal{A}_i^{\text{TV}}} \times A_{\mathcal{A}_i^{\text{TV}}} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ which returns the expected duration over all congestion bands:

$$c_i(s, e, t) = \begin{cases} \sum_j \mathit{cong}(r_i, e, j, t) \cdot \mathbb{E}[\rho_E(e, \iota^j)] & \text{if } s \in V \\ 0 & \text{otherwise,} \end{cases} \tag{6.6}$$

where $j \in [\![0, |I_e| - 1]\!]$, and a set of goal states G_i defined as:

$$G_i = \{v \in V \mid v = v_i^{\mathit{goal}}\}. \tag{6.7}$$

Time-varying transition functions increase model complexity [128, 129], making an exact TVMA-SSP solution impractical. To mitigate this, in Subsection 6.6.2 we present an approximate solution method, which adapts LRTDP [87] to incrementally build and solve an MDP abstraction of the TVMA.

6.6.2 LRTDP for TVMAs

To efficiently solve the TVMA-SSP for robot r_i (see Def. 6.6), we adapt LRTDP to TVMAs [87], which we refer to as *LRTDP-TVMA*. LRTDP is a trial-based heuristic tree search method that uses an admissible heuristic h_i to initialise the value function, so states unlikely to contribute to the optimal policy are not explored. LRTDP is presented in detail in Appendix B.

LRTDP-TVMA abstracts the TVMA behaviour into an MDP $\mathcal{M}_i$. $\mathcal{M}_i$ is built *incrementally*, as Bellman backups are only performed on states visited during a trial. An MDP state is a topological node paired with the time of arrival to that node. By including time in the state, we consider the time-varying nature of the transitions [91]. In the TVMA, continuous-time robot behaviour is captured by timed transitions, which we approximate in an MDP using expected action durations.

In MDP $\mathcal{M}_i$, the robot arrives at node v at time t, i.e. the state is (v, t), and chooses an edge e. There is then a congestion distribution on edge e, dependent on t. Similar to the TVMA, we capture this in the MDP transition function, where there is an MDP transition for each interaction band ι^j with probability $cong(r_i, e, j, t)$. The duration of e under interaction band ι^j is taken to be the expected value of the corresponding PTD, $\mathbb{E}[\rho_E(e, \iota^j)]$. For interaction band ι^j, the successor state in $\mathcal{M}_i$ is $(\texttt{trg}(e), t + \mathbb{E}[\rho_E(e, \iota^j)])$, i.e. the robot arrives at the destination of e at time $t + \mathbb{E}[\rho_E(e, \iota^j)]$.

In Alg. 6.1, we present LRTDP-TVMA, which incrementally builds the MDP abstraction of the TVMA. The main loop of LRTDP-TVMA (lines 3-5) runs trials until we reach convergence, or exceed a fixed computational budget. At the start of each LRTDP-TVMA trial, we set the MDP initial state to the robot's initial location at time 0 (line 10). For each trial, we sample through $\mathcal{M}_i$ until the termination condition in line 14 is met. By including time in the MDP state, a trial may never terminate, as we can keep executing actions to reach new non-goal states with higher times, e.g. by looping between two locations. To mitigate this, we introduce a time bound D, where any MDP state with a time greater than D is made a

Algorithm 6.1 LRTDP-TVMA (adapted from [37]).

Input: $\mathcal{A}_i^{\mathrm{TV}}$, v_i^{init}, admissible heuristic h_i, convergence threshold θ, time bound D
Output: Greedy policy π_i, value function V_{π_i}

function LRTDP-TVMA($\mathcal{A}_i^{\mathrm{TV}}, v_i^{init}, h_i, \theta, D$)
  $V_{\pi_i} \leftarrow h_i$
  while $(v_i^{init}, 0)$ not labelled solved and time left **do**
    LRTDP-TVMA-TRIAL($\mathcal{A}_i^{\mathrm{TV}}, v_i^{init}, \theta, D$)
  end while
end function

function LRTDP-TVMA-TRIAL($\mathcal{A}_i^{\mathrm{TV}}, v_i^{init}, \theta, D$)
  visited $\leftarrow$ empty stack
  $v \leftarrow v_i^{init}$, $t \leftarrow 0$
  while (v, t) not labelled solved **do**
    push (v, t) onto visited
    // **Check for termination**
    if $(v, t) \in G_i$ or $t > D$ **then**
      break
    end if
    // **Perform Bellman backup & update policy**
    $\pi_i(v, t) \leftarrow \mathrm{argmin}_{a \in A_{\mathcal{A}_i^{\mathrm{TV}}}} Q_{\pi_i}((v, t), a)$
    $V_{\pi_i}((v, t)) \leftarrow Q_{\pi_i}((v, t), \pi_i(v, t))$
    // **Sample successor MDP state**
    $\iota^j \leftarrow$ sample according to $cong(r_i, \pi_i(v, t), j, t)$
    $v \leftarrow \mathtt{trg}(\pi_i(v, t))$, $t \leftarrow t + \mathbb{E}[\rho_E(e, \iota^j)]$
  end while
  // **Update solved label for visited states**
  while visited $\neq$ empty stack **do**
    $(v, t) \leftarrow$ pop the top of visited
    if $\neg$CHECK-SOLVED$((v, t), \theta)$ **then**
      break
    end if
  end while
end function

dead end, terminating the trial. Trials are also terminated upon reaching the goal location. We modify the goal set G_i to ensure the goal is reached within time D:

$$G_i = \{(v, t) \in V \times \mathbb{R}_{\geq 0} \mid v = v_i^{goal} \text{ and } t < D\}. \tag{6.8}$$

If navigation always succeeds, robots reach their goal with probability 1. Under a finite horizon, the only way to miss the goal is to exceed time bound D. If D exceeds any realistic time to reach the goal, the planning model becomes a finite-horizon

SSP with *avoidable* dead ends. Goal states are defined as in Eq. 6.8, and any state with time greater than D is a dead end. A policy will reach the goal or a dead end, not both. As dead ends are avoidable, there exists a reachable goal state, and a policy that reaches it. Trials terminate at dead ends, and so all trials terminate. This allows LRTDP-TVMA to synthesise optimal policies, as policies that reach the goal do so in less time than those that reach dead ends [249].

In lines 18 and 19, we perform a Bellman backup on the current state, which updates the value function and policy. The value function V_{π_i} returns the expected cost to the goal from each state. The policy π_i specifies the action to execute in each state, where the actions in $\mathcal{M}_i$ are as in the TVMA. The Q value for a state/action pair is the immediate action cost, plus the value of subsequently following π_i to the goal:

$$Q_{\pi_i}((v,t),e) = c_i(v,e,t) \; + \sum_j cong(r_i,e,j,t) \cdot V_{\pi_i}((\texttt{trg}(e), t + \mathbb{E}[\rho_E(e,\iota^j)])), \quad (6.9)$$

where $j \in [\![0, |I_e| - 1]\!]$.

In lines 21-22, we sample a successor MDP state from the MDP transition function, which is defined using the congestion probabilities. First, we sample an interaction band given the policy edge and current time (line 21). The MDP successor is the destination of the policy edge after adding the expected edge duration to the previous time (line 22), where $\texttt{trg}(\pi_i(v,t))$ returns v if $\pi_i(v,t) = wait$.

In lines 25-30 we update the 'solved' label for states visited during the trial. A solved state need not be explored any further. Visiting a solved state terminates the trial (line 11). If the initial MDP state $(v_i^{init}, 0)$ is solved, LRTDP-TVMA has converged. The `CHECK-SOLVED` function is implemented as in [87], which we present in Appendix B. LRTDP-TVMA maintains a search tree, which forms the explored portion of the MDP abstraction of the TVMA. The `CHECK-SOLVED` function explores the sub-tree rooted at a state (v,t). If the value of (v,t) has changed below a threshold θ since its last Bellman backup, and all of its descendants in the sub-tree are solved, (v,t) is labelled as solved. If (v,t) is not solved, Bellman backups are performed on all unsolved states in the sub-tree.

The value $V_{\pi_i}((v,t))$ of MDP state (v,t) is initialised using an admissible heuristic $h_i : S_{\mathcal{M}_i} \to \mathbb{R}_{\geq 0}$ (line 2), which we define as:

$$h_i((v,t)) = \texttt{minCost}(v, v_i^{goal}), \tag{6.10}$$

where `minCost` returns the minimum travel cost between v and v_i^{goal}, which we compute with the Floyd-Warshall algorithm [250] (see Appendix A). The cost for edge e in `minCost` is $\mathbb{E}[\rho_E(e, \iota^0)]$, the expected edge duration under no congestion.

The output of LRTDP-TVMA is a policy defined over the states of the MDP abstraction of the TVMA, i.e. pairs of topological nodes and times. During execution, we need a policy which is defined for each point in continuous time, as robots will never arrive at a node at precisely the time in an MDP state. For this, we use the LRTDP-TVMA policy to create a piecewise constant policy [251], which for node v at time t returns the LRTDP-TVMA policy action for the MDP state at v with time closest to t. This creates a set of continuous-time intervals for each node v, where the policy action remains the same within each interval.

Example 6.2. Let π_i be a policy synthesised by LRTDP-TVMA (see Alg. 6.1), where for node v, $\pi_i(v,1) = a_1$ and $\pi_i(v,2) = a_2$. The piecewise constant policy π_i' created from π_i returns a_1 for node v before time 1.5 and a_2 after, i.e.:

$$\pi_i'(v,t) = \begin{cases} a_1 & t \in [0, 1.5) \\ a_2 & t \in [1.5, \infty). \end{cases} \tag{6.11}$$

LRTDP-TVMA quickly converges to a solution by avoiding the portions of the state space which do not contribute to the optimal policy. An alternative planning approach could be to apply UCT [90] to continuous space single-robot models, where each state considers a robot's location and time, similar to the MDP abstraction of the TVMA. Upon executing an edge action, there are infinitely many successor states, each with a different continuous-time value. The continuous transition distribution is a mixture of the duration distributions for that edge, weighted by the congestion distribution. UCT can be adapted to continuous state spaces using progressive widening techniques [252], which limit the number of successors visited from a state. By sampling continuous durations rather than assuming the expected

value, we would expect continuous UCT to synthesise more efficient behaviour given enough trials. However, by sampling continuous durations we visit many more unseen states per trial than with LRTDP-TVMA. For each unseen state, we compute congestion distributions for each action, which significantly increases the time per trial. Further, many trials are required for UCT to converge. In practice, we found continuous UCT to be cumbersome, and so use LRTDP-TVMA to scalably synthesise robot behaviour.

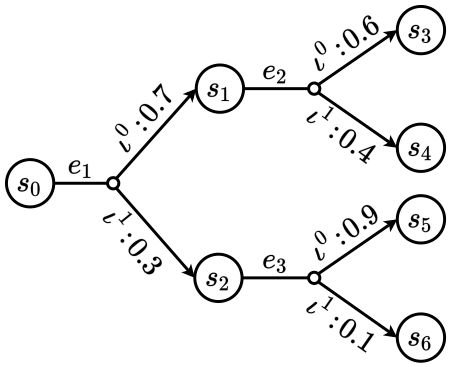
6.6.3 Constructing a Robot's Route CTMC

In the congestion-aware framework, robots take the best action considering those who planned before, whose effects are captured by the congestion probabilities and PRT. To compute congestion probabilities, the PRT requires a route CTMC for each robot that has planned (see Subsection 6.5.2).

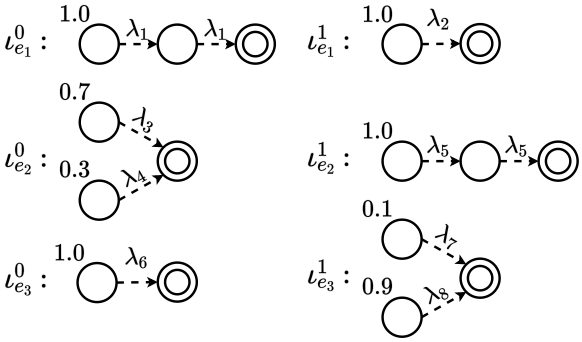
In this subsection, we describe how to construct route CTMCs using a robot's MDP produced by Alg. 6.1, its policy, and the PTDs. First, we apply the policy at each state in the robot's MDP to obtain the induced DTMC, which captures uncertainty over congestion. DTMC transitions branch according to the congestion probabilities, as in line 21 of Alg. 6.1.

Example 6.3. Consider Fig. 6.4a, which shows an induced DTMC annotated with actions and interaction bands. The DTMC transitions correspond to the congestion probabilities, i.e $\delta_{\mathcal{M}_i^{\pi_i}}(s_0, s_1) = cong(r_i, e_1, 0, t_{s_1}) = 0.7$, where t_{s_1} is the time in DTMC state s_1.

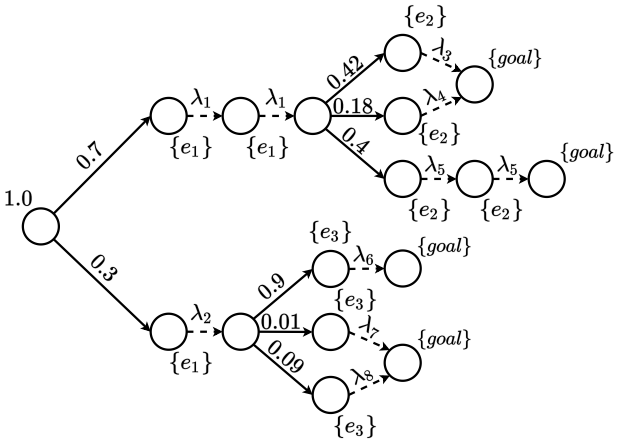
Next, we combine the DTMC and PTDs to produce a time-invariant MA without action choice $\mathcal{D}_i$, where states with immediate transitions have at most one action available [56]. There is a PTD for each interaction band a robot may observe on an edge. To build $\mathcal{D}_i$, we insert the corresponding PTD after branching by the interaction band, but before reaching the DTMC successor state. DTMC transitions are rewired to PTD initial states, as in Eq. 6.3. Transitions to the PTD absorbing state are rewired to the successor DTMC state. States in $\mathcal{D}_i$ and the route CTMC are labelled as in the robot's TVMA.



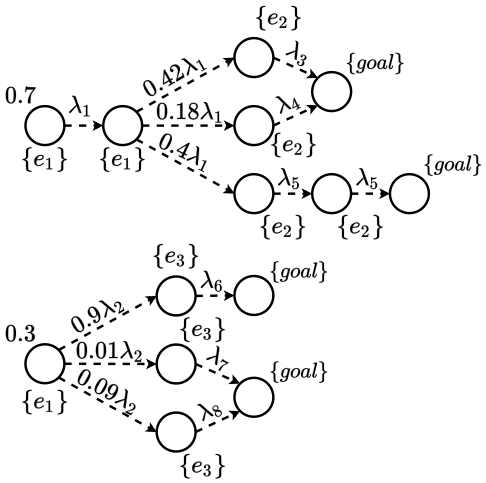
(a) An induced DTMC $\mathcal{M}_i^{\pi_i}$.



(b) PTDs experienced in the MDP/DTMC.



(c) The time-invariant MA without action choice $\mathcal{D}_i$ produced from the DTMC in Fig. 6.4a and PTDs in Fig. 6.4b.

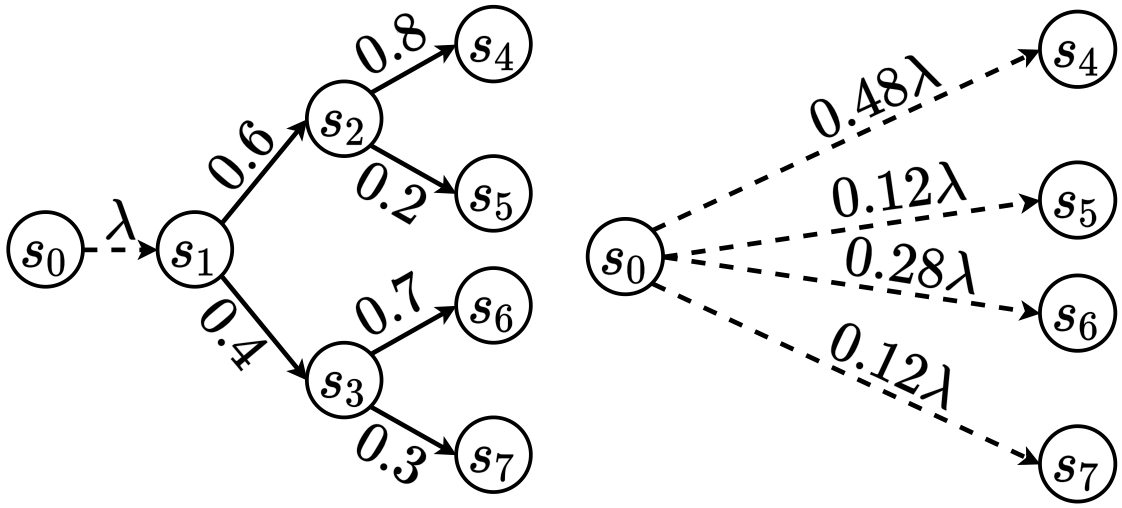


(d) The route CTMC $\mathcal{Q}_i$ produced by applying the method in Fig. 6.5 to MA $\mathcal{D}_i$ in Fig. 6.4c.

Figure 6.4: The process of generating a route CTMC from a robot's policy. Solid arrows are immediate/MDP transitions, dashed arrows are exponential transitions, and concentric circles represent PTD absorbing states.

Example 6.4. Consider Fig. 6.4a and Fig. 6.4b, where we assume the PTDs for the interaction bands in Fig. 6.4a are as in Fig. 6.4b. PTD absorbing states are represented by concentric circles. We present the resulting time-invariant MA without action choice $\mathcal{D}_i$ in Fig. 6.4c. To build $\mathcal{D}_i$, we insert the PTD for band $\iota^0_{e_2}$ between states s_1 and s_3 in Fig. 6.4a, adding transitions to the PTD initial states with probability $0.6 \cdot 0.7 = 0.42$ and $0.6 \cdot 0.3 = 0.18$ respectively. Transitions to the PTD absorbing state are rewired to s_3.

To build a route CTMC from a time-invariant MA without action choice, we collapse the immediate transitions into the exponential transitions. The collapsing



(a) A time-invariant Markov automata $\mathcal{D}_i$ without action choice.

(b) The CTMC $\mathcal{Q}_i$ obtained from collapsing the immediate transitions.

Figure 6.5: The process of collapsing immediate transitions into exponential transitions. Solid arrows are immediate transitions; dashed arrows are exponential transitions.

process is adapted from [121]. An exponential transition followed by a sequence of immediate transitions is collapsed into a single exponential transition via multiplication. The modified exponential transitions capture congestion uncertainty. Immediate transitions from the initial state are collapsed into the CTMC initial state distribution.

Example 6.5. Consider Fig. 6.5, which illustrates the collapsing process. For example, $\Delta_{\mathcal{Q}_i}(s_0, s_4) = \Delta_{\mathcal{D}_i}(s_0, s_1) \cdot \delta_{\mathcal{D}_i}(s_1, s_2) \cdot \delta_{\mathcal{D}_i}(s_2, s_4) = \lambda \cdot 0.6 \cdot 0.8 = 0.48\lambda$. If we apply this to the time-invariant Markov automata without action choice in Fig. 6.4c, we construct the route CTMC in Fig. 6.4d.

6.6.4 Complexity Analysis

The congestion-aware framework avoids the exponential complexity of planning on joint models such as the MRMA. For each robot, we obtain a policy with LRTDP-TVMA, and construct a route CTMC. A robot's MDP, built by LRTDP-TVMA, is a directed acyclic graph. Let $|A_i^{max}|$ be the maximum number of enabled actions across all states of $\mathcal{M}_i$. For each action, there are successor states for each interaction band. Each MDP transition advances time by at least $\min_{e,j} \mathbb{E}[\rho_E(e, \iota^j)]$,

and so the maximum MDP depth is $d = \left\lceil \frac{D}{\min_{e,j} \mathbb{E}[\rho_E(e,\iota^j)]} \right\rceil$, i.e. the maximum number of transitions before reaching the goal or a dead end. Therefore, the MDP state space is bounded by $\mathcal{O}((|A_i^{max}| \cdot |I_e^{\max}|)^d)$, where $I_e^{\max}$ is the largest interaction band set. Route CTMCs are constructed from induced DTMCs, where only one action is enabled at each state. Each DTMC transition is replaced with a PTD, and so route CTMCs have at most $\mathcal{O}(|S_{\mathcal{P}}^{\max}| \cdot |I_e^{\max}|^d)$ states, where $S_{\mathcal{P}}^{\max}$ is the largest PTD state space. The time complexity for route CTMC construction is equal to its size. Neither of these models depend on the number of robots.

For planning, we abstract a robot's TVMA into a finite-horizon SSP MDP, which can be solved with $|S_{\mathcal{M}_i}|$ backups, one for each state [37]. LRTDP improves upon this in practice. For the backup at state $s \in S_{\mathcal{M}_i}$, we compute the weighted sum of successor state values for each enabled action (see line 18, Alg. 6.1), which has time complexity $\mathcal{O}(|A_i^{max}| \cdot |S_{\mathcal{M}_i}|)$. Moreover, for each enabled action we must compute the corresponding congestion distribution, which requires the transient probabilities of up to $n-1$ route CTMCs. Let $S_{\mathcal{Q}}^{\max}$ be the largest route CTMC state space, and Λ be the maximum exit rate across all PTDs. CTMC transient analysis has a worst-case time complexity of $\mathcal{O}(|S_{\mathcal{Q}}^{\max}|^3 \cdot \log(\Lambda D))$ [40, 46], and so the time complexity for computing congestion distributions is $\mathcal{O}((n-1) \cdot |S_{\mathcal{Q}}^{\max}|^3 \cdot \log(\Lambda D))$. Therefore, single-robot planning has time complexity $\mathcal{O}(|S_{\mathcal{M}_i}| \cdot |A_i^{max}| \cdot (|S_{\mathcal{M}_i}| + (n-1) \cdot |S_{\mathcal{Q}}^{\max}|^3 \cdot \log(\Lambda D)))$, which scales polynomially in the team size. This result extends to planning for the whole team.

6.7 Policy Execution Model Refinement

In this section, we detail our approach to Problem 6.2, i.e. policy evaluation. Our goal is to evaluate single-robot properties given the policies synthesised during planning. Our approach leverages our solution to Problem 6.1, namely the use of route CTMCs to capture robot policy execution under congestion.

In Section 5.3, we presented CAMAS, which evaluates the asynchronous multi-robot behaviour induced by a set of policies by sampling through an MRMA and analysing the sample traces. Sampling overcomes the exponential complexity of the

Algorithm 6.2 The Iterative CTMC Refinement Procedure.

Input: PRT, set of policies $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$
Output: Refined set of CTMCs $\{\mathcal{Q}_1, \mathcal{Q}_2, ..., \mathcal{Q}_n\}$

1: **while** not converged **do**
2: $r_i \leftarrow$ `chooseNextRobot()`
3: $\mathcal{Q}_i \leftarrow$ `refineCTMC(`r_i`)`
4: $update(r_i, \mathcal{Q}_i)$
5: **end while**
6: **return** $\{\mathcal{Q}_1, \mathcal{Q}_2, ..., \mathcal{Q}_n\}$

MRMA, which prevents us from applying exact model checking techniques. CAMAS scales quadratically in the team size (see Section 5.4), and many samples are required for accurate policy evaluation. Further, for single-robot properties, CAMAS has to simulate the whole MRS, which may be cumbersome for large teams. Therefore, we consider approximating the MRMA using the single-robot models stored in the PRT. This provides a more compact representation of multi-robot behaviour, where we evaluate single-robot properties using model checking techniques.

Route CTMCs capture the effects of the other robots through congestion distributions. However, the *initial* route CTMC for r_i only considers robots r_1 to r_{i-1}, providing inaccurate congestion information. For policy evaluation, we require more accurate representations of congestion, and so in Alg. 6.2 we propose an iterative procedure that incrementally updates the route CTMCs after *all* robots have planned. In each iteration, a route CTMC is selected and refined using the current information in the PRT. The updated route CTMC is then inserted back into the PRT for the next iteration.

In line 2 of Alg. 6.2, we choose the next robot to update. In line 3, we refine the chosen robot's route CTMC as in Subsection 6.6.3, using the current congestion probabilities obtained from the PRT. The refined CTMC differs only in its congestion probabilities; its structure remains unchanged. Further, refining a robot's route CTMC does not change its policy. In line 4, we update the robot's CTMC in the PRT. Lines 2-4 are repeated until all models converge.

Choosing the Next Robot. The `chooseNextRobot` function in Alg. 6.2 may be defined arbitrarily under a fairness assumption, i.e. each robot is selected infinitely often. However, this choice affects the rate of convergence for Alg. 6.2. In Section 6.8, we compare a number of heuristics for choosing the next robot.

Testing for Convergence. To test for convergence in Alg. 6.2, we consider how a robot's route CTMC changes between updates. We introduce a distance function over CTMCs with the same state space, defined as the maximum absolute difference in the rate of a transition:

$$d(\mathcal{Q}, \mathcal{Q}') = \begin{cases} \max\limits_{s,s'} |\Delta_{\mathcal{Q}}(s, s') - \Delta_{\mathcal{Q}'}(s, s')| & \text{if } S_{\mathcal{Q}} = S_{\mathcal{Q}'} \\ \infty & \text{otherwise.} \end{cases} \tag{6.12}$$

Let $\mathcal{Q}_i$ be r_i's CTMC prior to its last refinement and $\mathcal{Q}'_i$ be r_i's current CTMC. The test for convergence is defined as:

$$\max_{r_i \in R} d(\mathcal{Q}_i, \mathcal{Q}'_i) < \xi, \tag{6.13}$$

where ξ is a small convergence threshold.

The iterative approach in Alg. 6.2 and CAMAS are two orthogonal approaches to multi-robot policy evaluation. The CTMCs produced by Alg. 6.2 are much smaller than the MRMA and so we can use exact model checking techniques for policy evaluation. However, these CTMCs are approximate. Further, the single-robot CTMCs cannot be used to evaluate global properties such as the probability of the makespan falling within a time bound. The MRMA is a more accurate model of multi-robot behaviour, where the single-robot CTMCs only capture the MRMA exactly for uncongested problems, where the MRMA is the parallel composition of the single-robot CTMCs, after resolving non-determinism. For all other problems, the single-robot CTMCs lose information over the MRMA, as the effects of congestion are summarised using only information local to robot r_i, whereas the MRMA directly observes congestion in the joint state. However, we cannot use exact policy evaluation methods on the MRMA due to its size. Therefore, CAMAS uses sampling for multi-robot policy evaluation. This introduces a trade off between model accuracy

and whether policies are evaluated exactly or approximately, where the choice of policy evaluation technique is problem dependent. For example, as congestion decreases, the MRMA gets closer to the parallel composition of the single-robot CTMCs, which therefore more closely approximate the MRMA. Consequently, model checking the single-robot CTMCs in low congestion problems may get closer to the true value of a property than CAMAS, dependent on the number of simulations. If we run enough CAMAS simulations we will get sufficiently close to the true property value, though this may be cumbersome compared to model checking the single-robot CTMCs. In the next section, we compare the single-robot CTMCs to the MRMA, and demonstrate that the refined single-robot CTMCs capture congestion more accurately than the initial CTMCs stored in the PRT after planning.

6.8 Experiments

In this section we analyse the congestion-aware framework in terms of its scalability, execution-time performance, and ability to evaluate the multi-robot behaviour induced by the synthesised policies. All experiments are run on Ubuntu 16.04, with an Intel Core i7-8700 CPU@3.2GHz, and 64GB of RAM. We use the PRISM model checker for CTMC transient analysis and model checking [48]. All software is written in Python, except for PRISM, which is written in Java/C++. All experimental problems follow Problem 6.1, i.e. synthesise a set of policies that send each robot to its goal location.

6.8.1 Experimental Setup

PRT Pruning

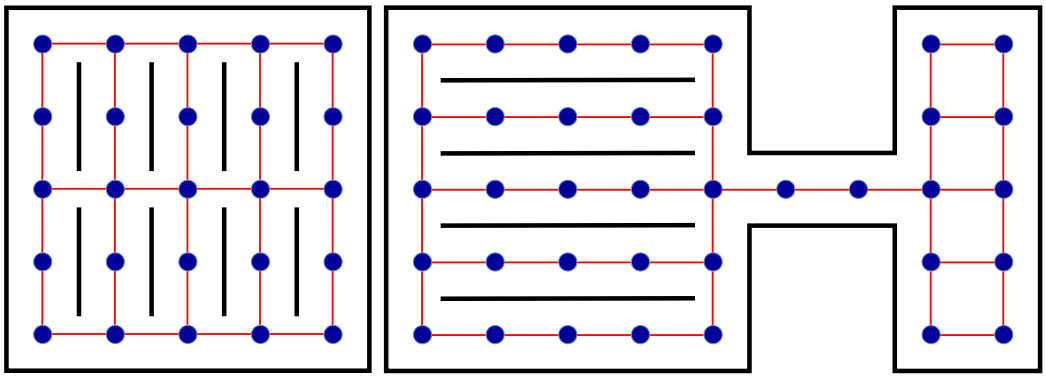
In the PRT, very small congestion probabilities are sometimes observed that bear no practical gain, but increase planning complexity. Therefore, we define a *pruning threshold* ε and set all congestion probabilities below ε to 0. The congestion distribution is then normalised to adjust for the removed probability mass. For all experiments in this chapter, $\varepsilon = 10^{-4}$.

Map Resource Groups

In this section, we assume bidirectional map resource groups. Formally, for edge $(v, v') \in E$, $g((v, v')) = \{(v, v'), (v', v)\}$, i.e. the edge and its reversed counterpart. For node $v \in V$, $g(v) = \{v\}$. By using bidirectional map resource groups alongside scalar contexts, we do not distinguish between the direction a robot travels between two nodes. Vector contexts would distinguish between the edges in each direction and improve model accuracy (see Section 5.5). However, there are more vector contexts per action, which increases the branching factor of our planning models and increases the complexity of planning, and so we do not use them here.

Experimental Baselines

This thesis is the first to introduce the congestion-aware planning problem, making it difficult to directly apply existing techniques. For example, we cannot use MAPF solvers, as the stochastic nature of the environment means collision avoidance cannot be guaranteed during execution. Moreover, we cannot compare against an optimal solution synthesised on the MRMA (see Chapter 5), as its size prevents the application of optimal planning methods. In this section, we consider the spectrum of techniques for handling execution-time robot interactions during planning, and compare against two baselines which represent either end of the spectrum. For all methods, execution-time robot interactions are resolved by a motion planner, not the high-level planner (see Subsection 6.8.3). The first baseline treats all robots independently, i.e. robots ignore each other and the effects of robot interactions during planning, and rely on a motion planner to resolve interactions during execution. Under this baseline robots follow their shortest path to the goal, assuming no congestion. The second baseline handles interactions conservatively by planning to avoid them. This baseline acts similarly to our framework: robots plan sequentially using the priority order heuristic in [107], and insert route CTMCs into the PRT after planning. However, under this baseline we only consider an edge during planning if the probability of one or more robots on the edge is less than a threshold $\zeta = 0.1$. If this condition is satisfied, the TVMA models the



(a) A 5×5 warehouse. **(b)** Warehouse with tunnel.

Figure 6.6: The maps used for the synthetic experiments.

robot deterministically traversing the edge under no congestion. The threshold ζ is set as low as possible while still allowing plans to be synthesised. This baseline is similar to a deterministic MAPF approach, as robots plan to almost entirely avoid congestion. If the threshold is set to 0, the baseline becomes a MAPF solver, where environmental uncertainty prevents plans from being found, as the collision avoidance constraint cannot be satisfied with probability 1. The congestion-aware planner lies in the middle of the solution spectrum by explicitly reasoning over congestion during planning. Under heavy congestion, the planner will try to route robots away from each other, whereas if there is little congestion robots will follow their shortest paths, similar to the independent baseline.

6.8.2 Validation in Synthetic Environments

We generated 3 maps using synthetic data: a 5×5 and 15×15 warehouse map (see Fig. 6.6a, the 15×15 map is the 5×5 map scaled up), and a warehouse with a tunnel, designed to force congestion (see Fig. 6.6b). The map sizes refer to the number of nodes along each dimension.

Synthetic duration distributions were created from lognormals, where the mode and variance increases with the number of robots, similar to what we observe in practice. PTDs were fit from 1000 lognormal samples using the method in [51] (see Appendix D). Recall that interaction bands over scalar contexts are subsets of $[\![0, n-1]\!]$ (see Subsection 6.5.1). We define four interaction bands for each map

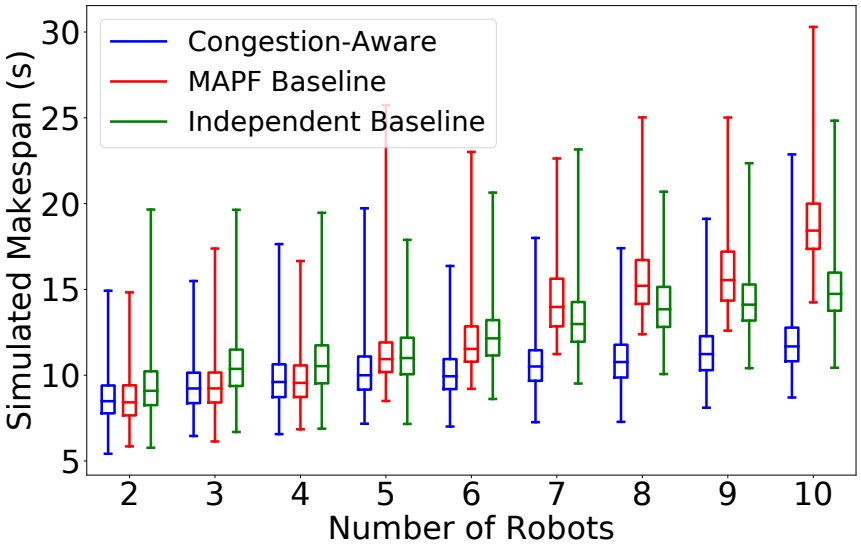


Figure 6.7: The simulated makespans for each of the 3 planning methods in the 5×5 warehouse map.

resource, $[\![0, 0]\!], [\![1, 3]\!], [\![4, 5]\!], [\![6, n-1]\!]$. All edges follow the same distributions, but with minor variations on each edge to ensure representative MDP state spaces in Alg. 6.1. LRTDP is run until convergence or 150 trials are run, where the MDP time bound $D = 200s$. The expected wait duration $\mathbb{E}[\mathcal{P}_\omega]$ is set to the expected edge duration (all edges share the same length) under no congestion.

Performance of the Obtained Policies

To assess planning performance, we generated a problem for 2-10 robots on the 5×5 warehouse map. Each additional robot increases the chance of congestion. For each problem, we measure execution-time team performance in terms of the makespan. We measure the makespan by running 1000 CAMAS simulations for each method/number of robots. Empirically we found that 1000 simulations provided an effective trade-off between computational cost and variance over the makespan. In Fig. 6.7, we present the simulated makespans, and in Table 6.1 we present the mean makespans alongside the results of a Mann-Whitney U statistical significance test, where $p = 0.05$.

Table 6.1: The mean makespans for the results in Fig. 6.7. Bold values have the lowest mean for that number of robots. Methods with highlighted cells do not produce statistically significantly higher makespans than the method with lowest mean, according to a one-sided Mann-Whitney U test with $p = 0.05$.

Number of Robots	Congestion-Aware	MAPF Baseline	Independent Baseline
2	8.64	**8.62**	9.31
3	**9.40**	9.41	10.57
4	9.87	**9.78**	10.75
5	**10.25**	11.21	11.20
6	**10.20**	12.06	12.31
7	**10.72**	14.40	13.24
8	**10.94**	15.57	14.08
9	**11.46**	15.97	14.37
10	**11.92**	18.80	14.97

Outliers occur in Fig. 6.7 when congestion is heavier than expected, or when durations are sampled near the PTD tails. For five robots and above, the congestion-aware planner statistically significantly outperforms both baselines. The MAPF baseline is conservative, synthesising long routes with high makespans. The independent baseline ignores robot interactions, causing heavy congestion during execution and increasing the makespan. Conversely, the congestion-aware planner distributes robots across the map, reducing the makespan. The makespan gap between methods is lower for smaller teams. For fewer robots there is less congestion, and so all three methods synthesise similar policies. In Subsection 6.8.3, we assess the planning methods in a more realistic robotic simulator.

Planner Scalability

To analyse the scalability of planning, 20 random 2-15 robot problem configurations were generated for each environment. We measure scalability in terms of the time for all robots to obtain a policy. This includes the time for route CTMC construction and congestion probability computation. Scalability results for the congestion-aware planner and baselines are shown in Fig. 6.8.

The congestion-aware framework mitigates the exponential complexity of joint space planning on all maps. Larger maps increase planning time, as robot routes

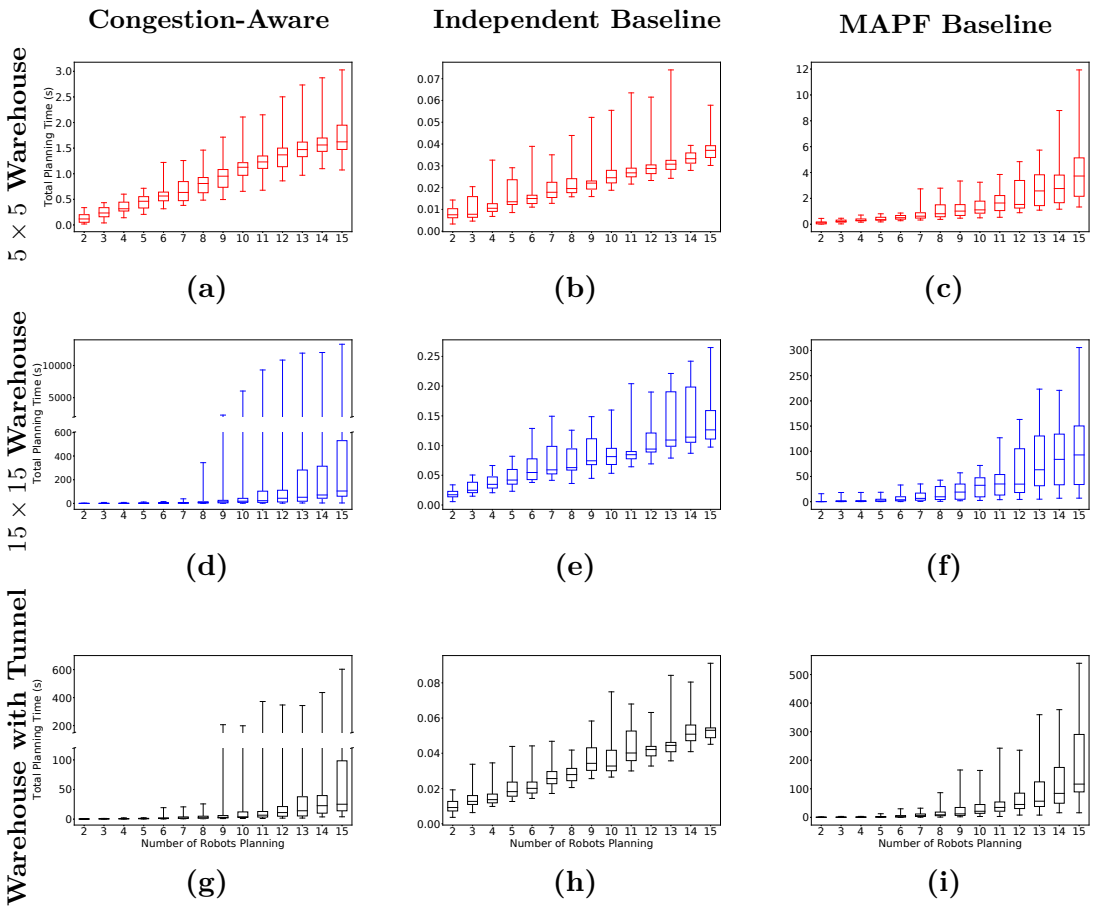


Figure 6.8: Planner scalability across the warehouse maps. Fig. 6.8a-6.8c show the scalability on the 5×5 warehouse. Fig. 6.8d-6.8f show the scalability on the 15×15 warehouse, and Fig. 6.8g-6.8i show the scalability on the warehouse with tunnel map.

are typically longer. The variance in scalability is due to the variance in congestion across problem configurations. If congestion is low, the MDP branching factor in Alg. 6.1 decreases, as fewer interaction bands are observed. This reduces planning time, as LRTDP-TVMA converges in fewer trials and computes fewer congestion probabilities. For example, it typically takes longer to plan on the warehouse with tunnel than the 5×5 warehouse despite their similar size, as the tunnel causes congestion. In the worst case, the congestion-aware framework takes a significant time to plan, as prolonged congestion drastically increases the size of the MDP abstraction of the TVMA. For example, in Fig. 6.8d and Fig. 6.8g, 1 of the 20 problems takes much longer to plan for, where for 15 robots, planning takes 13315 and 602 seconds respectively.

The independent baseline is significantly faster than the other methods, as planning reduces to an A* search over the topological map. The congestion-aware planner plans quicker than the MAPF baseline around 60% of the time. The constrained behaviour of the MAPF baseline causes robots to take longer routes, increasing the LRTDP-TVMA trial time. However, in the worst case, planning is significantly slower under the congestion-aware framework (cf. Fig. 6.8d). Under heavy congestion, the MDPs built by LRTDP-TVMA have high branching factors, increasing the state space. Computing congestion probabilities requires expensive model checking operations, which increase planning time. This state space increase is smaller under the MAPF baseline, where only one interaction band is observed per edge. Further, congestion causes the MAPF baseline to frequently use wait actions, which do not require costly PRT operations.

The Effect of Interaction Bands

Next, we analyse how the number of interaction bands per map resource affects planning performance and scalability. We generate a single 10 robot problem with heavy congestion on the warehouse with tunnel map in Fig. 6.6b. We vary the number of interaction bands between 2 and 10. For each number of bands, we run 20 repeats to measure the total planning time, and run 1000 CAMAS simulations to measure the makespan, where CAMAS uses an interaction band per scalar context, the most accurate representation of congestion. For this experiment, interaction bands are discrete intervals over $[\![0, n-1]\!]$ that are as equal in size as possible, apart from the first band, which is always $[\![0, 0]\!]$. When this was not possible, interaction bands were smaller for scalar contexts that represent lower numbers of robots. For example, for 5 interaction bands, the bands were set to $[\![0, 0]\!], [\![1, 2]\!], [\![3, 4]\!], [\![5, 6]\!], [\![7, 9]\!]$. We present the interaction band scalability and makespan results in Fig. 6.9 and Fig. 6.10 respectively.

Planning complexity increases with the number of interaction bands, as the MDP branching factor in Alg. 6.1 increases. For 8-10 bands, the rate of increase decreases, as the newly added bands are for scalar contexts that capture higher

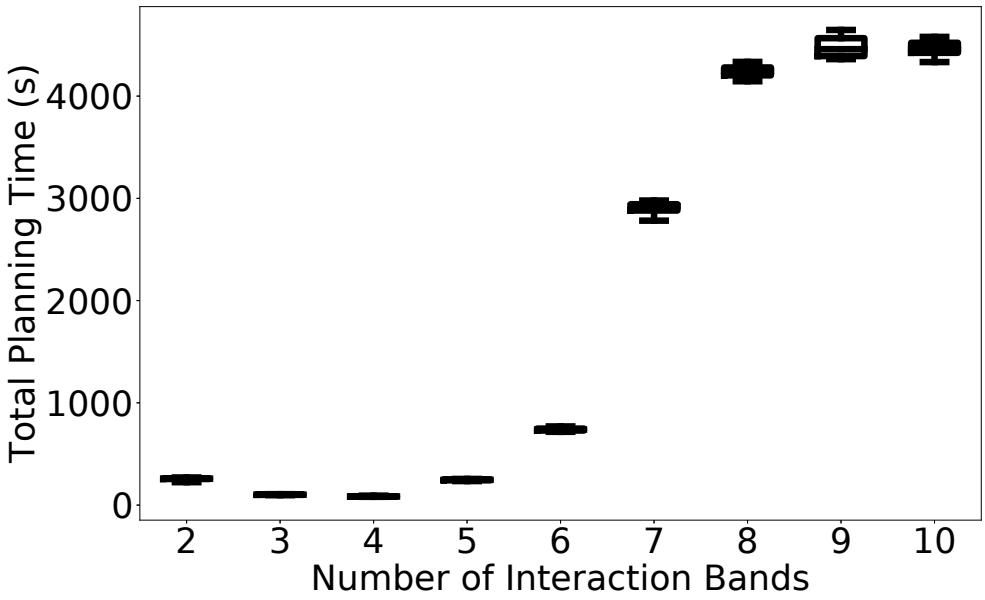


Figure 6.9: The effect the number of interaction bands have on planning time.

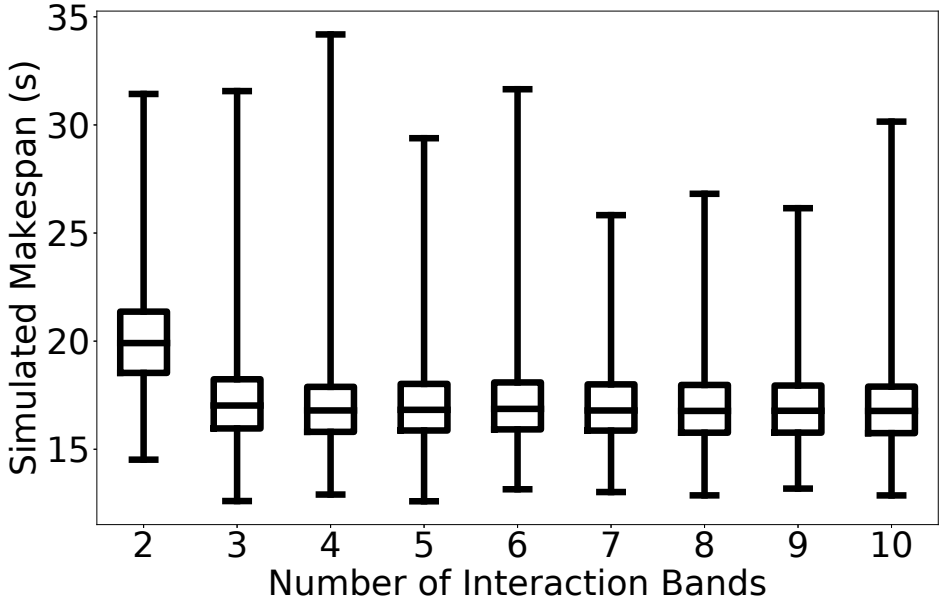


Figure 6.10: The effect the number of interaction bands have on makespan.

numbers of robots, which are rarely experienced. This has minimal effect on the MDP branching factor and planning time. The planning time for 2 and 3 bands is higher than for 4. If there are too few interaction bands, we plan on an inaccurate model of congestion. For example, for 2 interaction bands, the effects of 1 robot are treated the same as 9. Therefore, low congestion levels incur a large cost during planning, and so robots plan to avoid congestion, which increases the length of routes and LRTDP-TVMA trials, increasing the planning time. Longer routes increase the makespan, as seen in Fig. 6.10, where the average makespan for 2-3 bands is higher than for 4-10. This demonstrates the consequences of inaccurate congestion models. It is significantly faster to plan for 4 bands than for 10, despite no significant change in makespan, as robots are planning near identical routes. Therefore, using more interaction bands may increase planning time at no practical gain, and so we can synthesise efficient behaviour with relatively few interaction bands.

Analysing the Iterative Refinement Procedure

For each problem on the warehouse with tunnel map (see Fig. 6.6b) used to assess planner scalability, we ran Alg. 6.2 to assess its scalability and performance. The convergence threshold ξ was set to $\xi = 10^{-6}$. As discussed in Section 6.7, the CTMC refinement order in Alg. 6.2 affects convergence speed. We compare 3 different heuristics:

- *Random*: The next robot is chosen randomly.
- *Sequential*: Robots are selected in a round-robin fashion according to the planning priority order.
- *Maximum difference*: We choose the robot whose CTMC changed the most the last time it was refined, according to the difference metric in Eq. 6.12. To compute this, all CTMCs must be refined at least once. Therefore, for the first n iterations we use the sequential heuristic. The CTMC difference acts as a proxy for how inaccurate a robot's CTMC is. This is similar to prioritised sweeping for ordering dynamic programming updates [253].

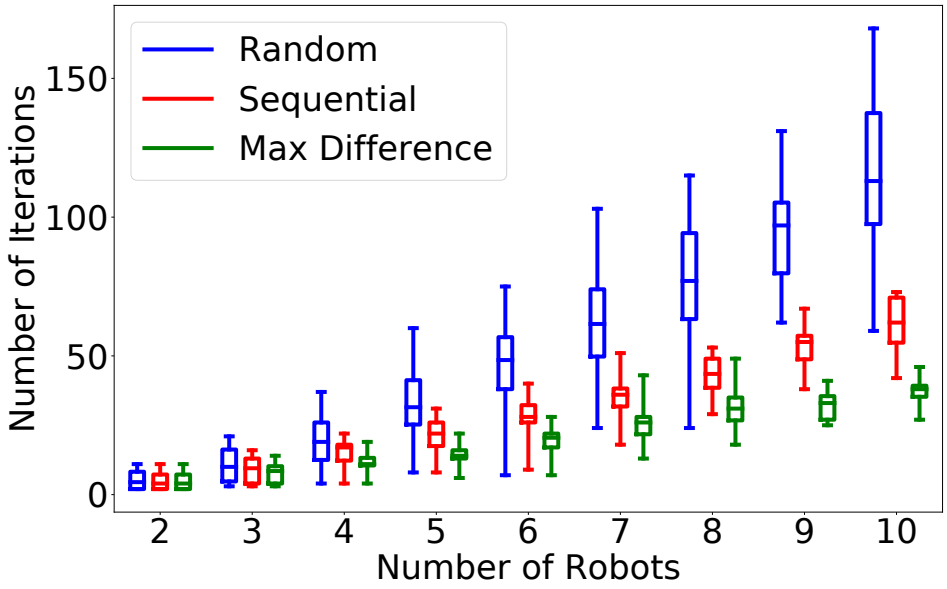


Figure 6.11: The number of Alg. 6.2 iterations for each heuristic to reach convergence.

The above heuristics operate under a fairness assumption. However, the use of a convergence threshold breaks this assumption, as each CTMC is only refined a finite number of times. This may affect the results of Alg. 6.2 under different heuristics.

In Fig. 6.11, we present the number of iterations/CTMC refinements to reach convergence in Alg. 6.2. Moreover, we present the mean convergence times in Table 6.2. The random heuristic scales poorly, as it often refines robot CTMCs who have no new information to incorporate, e.g. if that robot's route is clear of congestion. The maximum difference heuristic scales best in terms of iterations and time, as it only rebuilds CTMCs likely to be improved. The sequential heuristic scales worse than the maximum difference heuristic, as the planning order forces the refinement of CTMCs that have already converged. The worst case convergence times highlight the difference between heuristics. For 10 robots, the worst case time for the sequential and random heuristic were approximately 2.6 and 4.6 times longer than the maximum difference heuristic respectively.

Though the MRMA cannot be replicated with single-robot CTMCs, Alg. 6.2 is designed to provide a close, tractable approximation. Therefore, the final CTMCs produced by Alg. 6.2 should more accurately capture the MRMA than the initial

Table 6.2: The mean times in seconds ± standard deviation for each of the 3 heuristics to complete Alg. 6.2. Bold values have the lowest mean for that number of robots.

Number of Robots	**Random Heuristic**	**Sequential Heuristic**	**Max Difference Heuristic**
2	1.34 ± 2.05	$\mathbf{1.29 \pm 1.99}$	$\mathbf{1.29 \pm 1.98}$
3	3.78 ± 5.2	2.98 ± 3.85	$\mathbf{2.94 \pm 3.83}$
4	8.65 ± 11.9	6.3 ± 7.96	$\mathbf{5.75 \pm 6.96}$
5	38.1 ± 89.5	20.7 ± 48.6	$\mathbf{16.3 \pm 33.7}$
6	152 ± 270	79.7 ± 132	$\mathbf{73.3 \pm 126}$
7	236 ± 451	161 ± 343	$\mathbf{151 \pm 321}$
8	706 ± 1582	390 ± 903	$\mathbf{352 \pm 874}$
9	3018 ± 6967	1776 ± 4187	$\mathbf{877 \pm 2029}$
10	7747 ± 18807	3948 ± 10370	$\mathbf{1693 \pm 4067}$

CTMCs stored in the PRT immediately after planning. To demonstrate this, we evaluate expected time and time-bounded reachability properties on the initial and final CTMCs, as well as on the MRMA. We then consider the mean error between the single-robot CTMC and MRMA values across the team. It is intractable to apply exact model checking techniques to the MRMA due to its size. Therefore, we evaluate properties on the MRMA by running 1000 CAMAS simulations. Though this is approximate, by running enough simulations we get sufficiently close to the true value of the property. For time-bounded reachability, the time bound is the expected traversal time, computed from the PTDs, for the longest straight line of nodes in the map, under no congestion. We present the results for expected time and time-bounded reachability in Fig. 6.12 and Fig. 6.13 respectively.

For both properties, the final CTMCs consistently give a small error against the MRMA, which is lower than the error from the initial CTMCs in the majority of cases. The initial and final CTMCs produce the same error for uncongested problems, i.e. the robots never interact. The initial CTMCs produce a smaller error for problems where 2 or more robots traverse the same edges simultaneously. There, the first robot plans optimistically, assuming no other robots are present. Therefore, its initial route CTMC ignores the congestion that will occur during execution, and underestimates the time to reach the goal. Compared to the value obtained on the final CTMC, this underestimate is closer to the true MRMA

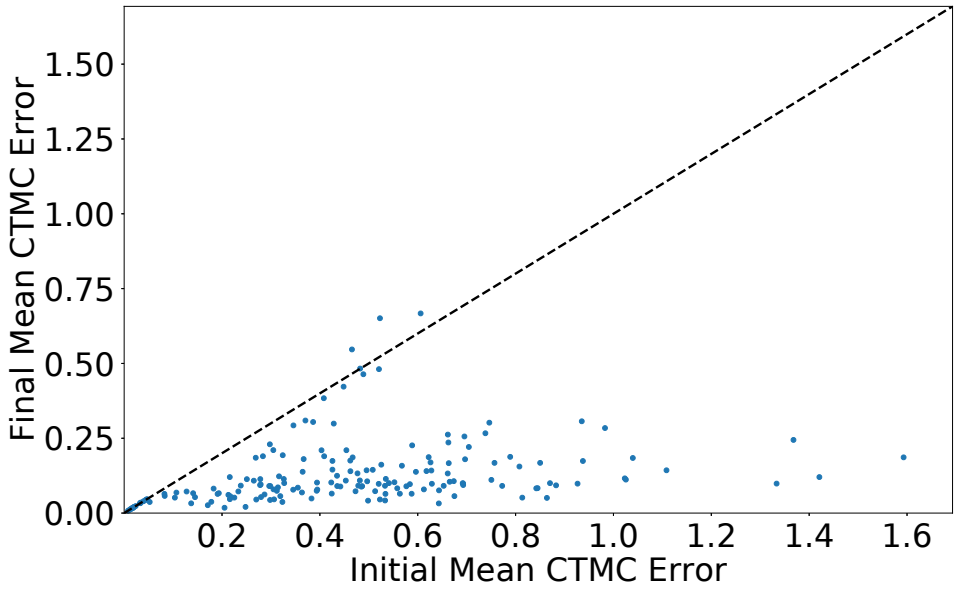


Figure 6.12: The difference in expected time error between the initial and final CTMCs. Each point gives the mean error across the entire team.

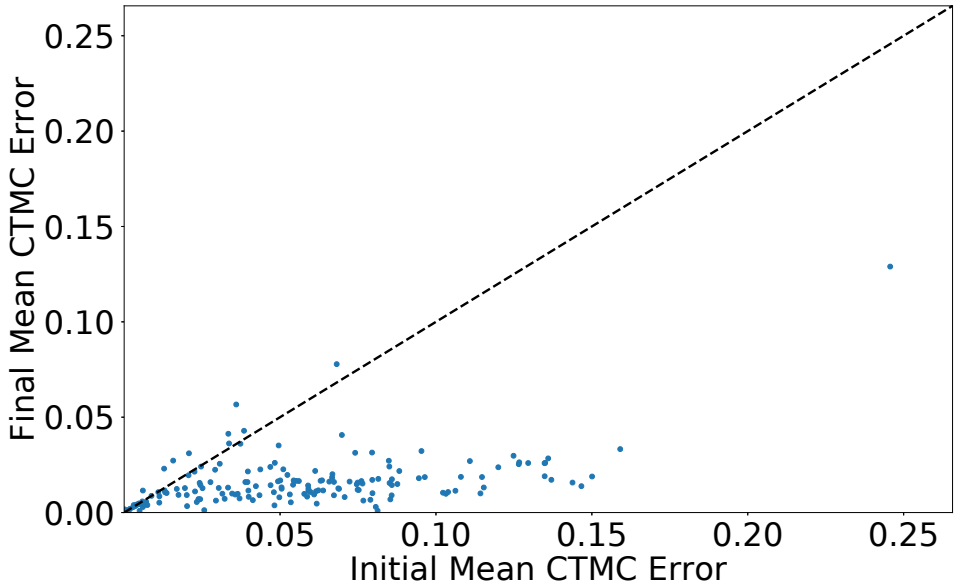


Figure 6.13: The difference in time-bounded reachability error between the initial and final CTMCs. Each point gives the mean error across the entire team.

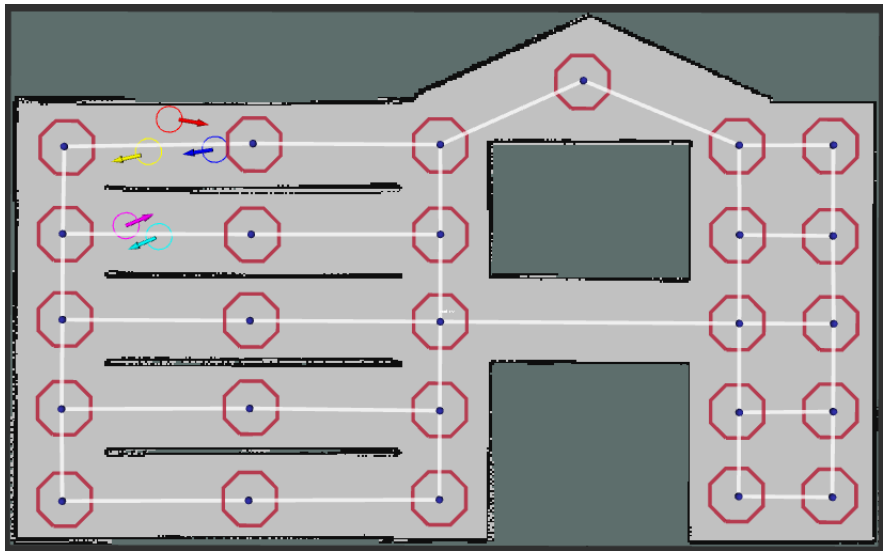


Figure 6.14: A congested warehouse environment with a topological map overlaid. The circles with arrows show robots in the map.

value, producing a smaller error. From these results, we demonstrate that the single-robot CTMCs produced by Alg. 6.2 closely approximate the MRMA, and can be used to accurately evaluate timed properties of the multi-robot behaviour induced by the synthesised policies.

6.8.3 Validation in a Simulated Environment

To assess execution-time performance, we simulate 5 robots in the warehouse in Fig. 6.14 using the Stage ROS simulator [239]. Robots navigate using Smach state machines and Move Base Flex tailored to multi-robot systems [247]. The state machines constrain robots to navigate via topological edges and maintain collision avoidance. A central node detects intersecting robot paths and instructs some robots to wait to avoid collisions. When a wait action is executed, a duration is sampled from $\mathcal{P}_\omega$. The two tunnels in Fig. 6.14 allow robots to take the longer tunnel if the shorter one is too congested. PTDs were fit using the EM algorithm in [51] (see Appendix D) from data collected in the simulator using edge partitioning (see Section 4.4). Fig. 4.7 shows an example set of PTDs from this map, alongside the data collected. We use 5 interaction bands, one for each scalar context, i.e. number of other robots.

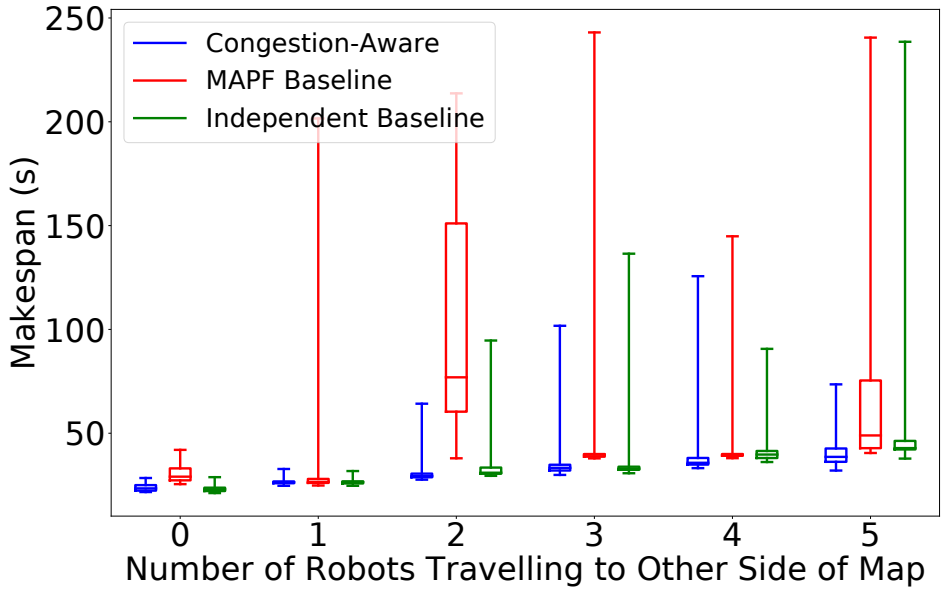


Figure 6.15: The observed makespans for each of the 3 methods in the Stage simulation.

The environment in Fig. 6.14 is only used for the simulation experiments in this subsection, and not for any of the experiments in synthetic environments, or in other chapters. These experiments were run before those in other chapters, where we scale up to larger environments. Due to the time required to run robot simulations, we designed Fig. 6.14 as a small environment to demonstrate congestion. We would expect strong planning performance on synthetic environments that use the map in Fig. 6.14, similar to Subsection 6.8.2.

Performance of the Obtained Policies

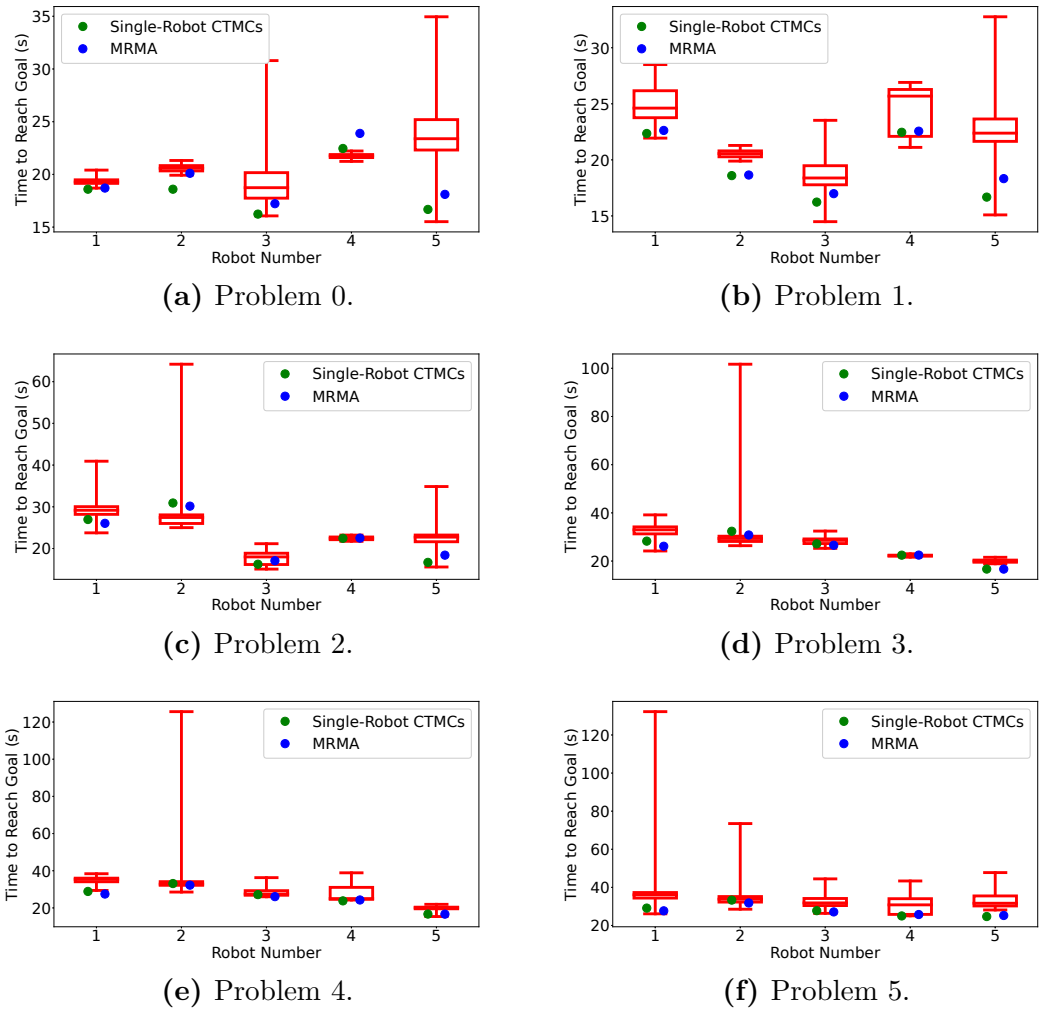
We consider 6 problem configurations to assess the congestion-aware planner and baselines. We run 40 repeats for each problem and method. We set the expected waiting time $\mathbb{E}[\mathcal{P}_\omega]$ to the expected duration of the shortest edge, under no congestion. In each problem, 3 robots start on the larger side of the map and 2 start on the smaller side. For problem 0, each robot's goal remains on the same side as their initial location. For problems 1-5, the problem number states how many robots' goals are on the other side of the map. Congestion increases as more robots travel to the other side of the map. To assess performance, we measure the makespan.

Table 6.3: The p-values from a one-sided Mann-Whitney U test for the makespans in Fig. 6.15. Bold values are those where the congestion-aware planner produces statistically significantly lower makespans than the relevant baseline, where $p = 0.05$

Problem Number	**Against Independent Baseline**	**Against MAPF Baseline**
0	8.92×10^{-1}	$\mathbf{5.04 \times 10^{-12}}$
1	3.48×10^{-1}	$\mathbf{3.27 \times 10^{-2}}$
2	$\mathbf{2.41 \times 10^{-6}}$	$\mathbf{1.42 \times 10^{-12}}$
3	6.59×10^{-1}	$\mathbf{2.42 \times 10^{-10}}$
4	$\mathbf{1.26 \times 10^{-5}}$	$\mathbf{1.29 \times 10^{-6}}$
5	$\mathbf{4.42 \times 10^{-5}}$	$\mathbf{2.73 \times 10^{-8}}$

We present the observed makespans in Fig. 6.15. Further, in Table 6.3, we present the results of a one-side Mann-Whitney U statistical significance test comparing the methods, where $p = 0.05$. Outliers in Fig. 6.15 occur when the motion planner takes longer to resolve congestion. The congestion-aware method outperforms the MAPF baseline in all problems, which avoids congestion regardless of the cost incurred. The stochastic nature of the environment means congestion may still occur under the MAPF baseline. This significantly affects the makespan, as congestion was unexpected during planning.

For less congested problems, the congestion-aware planner and independent baseline synthesise similar policies. For problems 0, 1 and 3, there is no statistical difference between them, as the shortest path is the lowest cost route. However, the congestion-aware planner outperforms the independent baseline in problem 2. This is likely due to the baseline randomly selecting between two identical length routes, where one causes congestion. The congestion-aware method will always choose the uncongested route. The congestion-aware method significantly outperforms the independent baseline in problems 4 and 5, as it utilises the top tunnel in Fig. 6.14 to minimise congestion. The independent baseline routes all robots through the central tunnel, producing heavy congestion. This demonstrates the independent baseline's inability to handle robot interactions.



(a) Problem 0. **(b)** Problem 1.

(c) Problem 2. **(d)** Problem 3.

(e) Problem 4. **(f)** Problem 5.

Figure 6.16: The time taken for each robot to reach its goal in simulation. Green circles show the expected time computed on the single-robot CTMCs produced by Alg. 6.2. Blue circles show the expected time computed on the MRMA using CAMAS. The problem number indicates how many robots travelled through a tunnel in Fig. 6.14.

Accuracy of Policy Evaluation

The refined single-robot CTMCs (and the MRMA) rely on the context-dependent PTDs accurately capturing execution-time behaviour. To validate our models, we run Alg. 6.2 using the congestion-aware policies synthesised in the previous experiment. We then evaluate the expected time property on each robot's CTMC, and the MRMA. We evaluate on the MRMA by averaging over 1000 CAMAS simulations. In Fig. 6.16, we compare these values against the times recorded over 40 simulation runs for each robot.

The single-robot CTMCs and MRMA typically underestimate the simulation times, suggesting some congestion effects are unmodelled. The congestion-aware framework relies on modelling assumptions such as the use of interaction bands and PTDs to model navigation durations. During planning, we compute congestion at the point a robot joins an edge. However, robots may join or leave the edge at a different point during execution. Further, in this chapter we assume scalar contexts and bidirectional map resource groups, and use edge partitioning for data collection, which produces less accurate duration models (see Section 5.5). We also do not consider the proximity of other robots on an edge. Despite this, the predicted expected times closely approximate those observed in simulation. This correlates with Section 5.5, where we demonstrated that the MRMA accurately captures multi-robot behaviour. This validates how the PTDs and congestion distributions effectively approximate congestion.

6.9 Discussion

The congestion-aware planner effectively routes robots under congestion, but assumes that a motion planner can always resolve congestion on an edge. This assumption is broken in very narrow environments such as fruit fields [4, 6], where aisles can only fit a single robot. In these environments, we want to avoid congestion, and so the MAPF baseline may be more appropriate. The MAPF baseline does not prevent congestion entirely though, and so a better approach may be to transform the topological map into a grid and apply traditional MAPF solvers, where robots execute synchronously. The independent baseline is highly scalable, and performs similarly to the congestion-aware planner on problems with little to no congestion. However, for these problems the congestion-aware planner matches the scalability of the independent baseline, as very few route CTMC model checking operations are required during planning.

In Section 6.8, we demonstrated how relatively few interaction bands are required to synthesise efficient robot behaviour. In that experiment, using two interaction bands decreased performance and increased planning time, due to inaccurate

modelling. However, using two interaction bands may not always have this effect. For example, the most accurate model for a two robot system that uses scalar contexts is to have one interaction band per context. The optimal number of interaction bands is dependent on the problem and environment. Further, there are multiple ways to specify a given number of interaction bands, where this specification affects model accuracy. For example, two interaction bands over the scalar contexts $[\![0, 9]\!]$ could be defined as $[\![0, 0]\!]$ and $[\![1, 9]\!]$, or $[\![0, 5]\!]$ and $[\![6, 9]\!]$. In this thesis, we design all interaction bands manually from empirical observations, and leave learning optimal interaction bands to future work.

After refinement, the route CTMCs in the PRT closely approximate the MRMA. When comparing CAMAS (see Section 5.3) to the iterative refinement approach in Section 6.7, there is a trade off between applying approximate methods on exact models, and applying exact methods on approximate models. We must also consider the time taken to refine the single-robot CTMCs using Alg. 6.2. For large MRSs, refinement takes a significant time which could be used to run CAMAS simulations. Therefore, the iterative refinement approach to **Q4 - Policy Evaluation** is most suitable for problems with small MRSs where congestion is infrequent (see Section 6.7). Further, though the refined single-robot CTMCs closely approximate the MRMA empirically, we cannot bound the error between these models, which could be arbitrarily large.

6.10 Summary

The MRMA in Chapter 5 scales exponentially with the number of robots, making exact planning methods intractable for anything more than small two-robot problems. Therefore, in this chapter we presented a congestion-aware framework for multi-robot planning under temporal uncertainty, which was introduced in [23], and expanded in [24]. Our framework approximates the MRMA with single-robot TVMAs that capture distributions over congestion. Congestion distributions are computed by applying model checking techniques to CTMCs describing robot policy execution, which are stored in a PRT. We synthesise policies that are robust to

unexpected execution-time delays using LRTDP-TVMA, which abstracts TVMAs into MDPs. After planning, we iteratively refine the route CTMCs in the PRT to evaluate the multi-robot behaviour induced by the synthesised policies. As the single-robot CTMCs in the PRT are smaller than the MRMA, we can apply exact model checking techniques to evaluate robot behaviour, rather than the sample-based approach in Chapter 5. The iterative refinement procedure first appeared in [24]. The congestion-aware framework outperforms baselines which ignore robot interactions during planning, or plan conservatively to avoid them. Moreover, after applying the iterative refinement procedure, the route CTMCs in the PRT closely approximate the MRMA. Though we have focused on multi-robot navigation tasks, our framework can be adapted to any multi-robot planning problem under temporal uncertainty. The congestion-aware framework contributes towards **Q2 - Modelling**, **Q3 - Planning**, and **Q4 - Policy Evaluation**. However, each robot has a local goal which we optimise for separately. In the next chapter, we use MRTA techniques for coordinating robots towards a global goal.

7

Proactive Multi-Robot Task Allocation Under Spatiotemporal Uncertainty

Contents

7.1 Introduction 140
7.2 Problem Formulation 143
7.2.1 Environment Model 143
7.2.2 Tasks and Schedules 144
7.2.3 Task Announcement Model 145
7.2.4 MRTA Under Spatiotemporal Uncertainty 148
7.3 Proactive SSI Auctioning Under Spatiotemporal Uncertainty 149
7.3.1 Auctioning Framework 150
7.3.2 Computing Bids 153
7.3.3 IWPs 156
7.3.4 Complexity Analysis 157
7.4 Experiments 159
7.4.1 Experimental Setup 159
7.4.2 Measuring Performance as Robots Increase 162
7.4.3 Measuring Performance as Tasks Increase 166
7.4.4 Schedule Cost Scalability 167
7.5 Discussion 168
7.6 Summary 169

7.1 Introduction

The congestion-aware planner in Chapter 6 solves multi-robot planning problems where each robot minimises the time to reach its goal location. In these problems, coordination decreases the time to reach the goal by reducing congestion, but is not required to get to the goal. This is demonstrated by the independent baseline in Section 6.8, where robots ignore each other during planning, and follow their shortest path to the goal. Though robots will always arrive at their goal locations there is no coordination, which increases congestion and the time cost. However, many problems require explicit coordination between robots to achieve a global goal. For example, autonomous taxis should coordinate to ensure a vehicle always reaches a customer within a fixed time [254]. MRTA techniques facilitate global coordination by assigning tasks to robots [7] (see Section 3.3).

MRTA algorithms have been used for teams of mobile robots to efficiently fulfil orders in warehouses [255], assist in healthcare facilities [208], and search for survivors in disaster zones [219]. Further, task allocation techniques have been applied to human teams to optimise fruit picking [256], warehouse operations [257], and taxi driver assignment [258]. In this chapter, we consider MRTA problems where mobile robots are assigned tasks to support humans in the environment, who are out of our control. In these problems, tasks correspond to assisting a single human, where tasks are *announced* at different locations during execution which often map to a human's location. Upon announcement, a robot must navigate to the task location and *service* it, i.e. execute the actions required by the task. However, for these problems, the announcement time and location of a task are often stochastic, due to disturbances from the environment or agents out of our control, such as order processing delays in a fulfilment centre.

Example 7.1. Consider Fig. 7.1, where robots operate alongside human pickers in a fruit field [4, 6]. Pickers fill up baskets with fruit as they traverse the field, and request a robot once their basket is full. The robot collects the basket and provides the picker with an empty one. The picker is unable to continue work until they



Figure 7.1: Robots operating alongside fruit pickers in a fruit field.

receive the new basket. This can be formulated as an MRTA problem, where each picker's request is a new task, and announcement occurs upon the picker making the request. Prior to announcement, the time and location of the picker's request is uncertain, due to factors such as human work rate and the yield of fruit in the field. The time and location of the request are also coupled, as a long announcement time suggests a picker has travelled further before requesting a robot.

Existing MRTA solutions either assume that all task announcements are known *a priori* [259], which is often unrealistic, or reactively allocate tasks after they are announced [225, 229]. However, reactive allocation yields long *waiting durations*, i.e. there is a large gap between a task's announcement time and the time a robot begins to service it. To reduce waiting durations, tasks should be allocated *proactively*, such that robots can navigate towards potential task locations prior to announcement. To achieve this, we must explicitly model the spatiotemporal processes that govern task announcement, i.e. for Example 7.1 we must model the progress of the fruit pickers.

In this chapter, we address **Q5 - Task Allocation** by presenting an MRTA framework for tasks with stochastic announcements, both in space and in time, that minimises the *expected total waiting duration* across all tasks. Though task announcement is stochastic, we assume that the tasks are known, where every task is eventually announced. This assumption holds in Example 7.1, as at any time there is one basket collection task per picker, and each picker's basket eventually fills

up, which triggers task announcement. Though we focus on problems where mobile robots assist humans out of our control, our framework can solve any MRTA problem that satisfies the above assumption on stochastic task announcement. We model task announcement by constructing a CTMC for each task, and apply model checking techniques to reason over announcement times and locations [40]. CTMCs can be fit from empirical data, and allow the processes governing announcement time and announcement location to be coupled, while also allowing for arbitrary distribution shapes [51]. These CTMC models expand our view of temporal uncertainty to include sources which do not directly affect robot action durations, and contribute towards **Q2 - Modelling**.

The MRTA problem in this chapter can be formulated as a simultaneous task allocation and planning (STAP) problem, similar to [98, 236] (see Subsection 3.3.5). STAP solutions construct a joint model where robots select tasks and the actions required to complete them. The STAP model can then be solved using standard techniques to synthesise a task allocation and a plan for each task. In [98, 236], robots are considered sequentially, i.e. robot one selects its tasks and synthesises plans for them, and then the model switches to robot two. However, sequential STAP models may be unsuitable for tasks with stochastic announcements, as its unclear how to capture the announcement processes alongside robot switching in the model. For MRTA under stochastic task announcement, we can extend the MRMA (see Chapter 5) to include immediate transitions representing task selection, and to explicitly model the evolution of the CTMC models of task announcement, i.e. the MRMA state is now the joint state of the robots and task models. Existing techniques for STAP under uncertainty can scale to around eight robots [98]. For the congestion-aware planning problem in Chapter 6, the MRMA cannot scale beyond small two robot problems (see Section 6.1). By extending the MRMA for STAP under stochastic task announcements, the model size increases further. In particular, the extended MRMA scales exponentially in the number of tasks, as the joint MRMA state now considers the states of the task announcement models. Therefore, applying STAP techniques to the MRMA is unsuitable for the MRTA

problem in this chapter. To mitigate the poor scalability of STAP on the MRMA, we decouple task allocation and planning, which allows for more scalable solutions, but decreases solution quality [14, 98].

In this chapter, we present a framework which extends sequential single-item (SSI) auctioning [200] for proactive task allocation, where each potential task location is auctioned separately. With this, we assign different robots to the same task, dependent on the announcement location. To handle uncertain task locations, robots navigate to *intermediate waiting points (IWPs)* which are near potential task locations, and wait for task announcement. IWPs are computed using task location distributions to minimise the expected waiting duration. Moreover, we reauction tasks online upon task updates. With this, we synthesise solutions that are robust to early announcements and unexpected delays. Though we focus on SSI, our framework can be adapted to use other MRTA algorithms in a straightforward way. We demonstrate the efficacy of our approach empirically using CAMAS, where we compare against a non-proactive approach which is representative of techniques common to the literature, and approaches which modify different components of our framework. The work in this chapter appears in [27].

7.2 Problem Formulation

In this section, we propose CTMCs as models for stochastic task announcement on a context-aware topological map, and define the problem of MRTA under spatiotemporal uncertainty.

7.2.1 Environment Model

The MRTA problem is more complex than the multi-robot planning problem as we must consider the joint space of robot tasks and plans, i.e. which tasks should a robot execute, and how should they execute them? Therefore, MRTA algorithms often use simplified environment models for scalability. In this chapter, we simplify the environment model by ignoring the context, i.e. ignoring the presence of other robots during navigation. Formally, we represent the environment

as a context-aware topological map $\mathcal{T}_C = \langle V, E, C, \rho_E \rangle$, as defined in Section 4.3, but where each topological edge $e \in E$ has only a single duration distribution, which we denote with $\rho_E(e)$. To compensate for these simplifications, we use richer multi-robot models during planning to synthesise robust behaviour for each task after obtaining an allocation. For example, the congestion-aware planner in Chapter 6 can be used to reason over congestion, which we will integrate with our MRTA framework in future work.

7.2.2 Tasks and Schedules

MRTA solutions assign a set of tasks $T = \{\tau^1, ..., \tau^m\}$ to a set of robots $R = \{r_1, ..., r_n\}$ to minimise cost.

Definition 7.1. A *task* $\tau \in T$ is announced at a topological node $v \in V$. To complete a task, a robot must navigate to the task location and service it, i.e. execute the task's required actions. Tasks cannot be serviced before announcement.

For simplicity of presentation, we assume that tasks are serviced instantaneously upon a robot arriving to the task location. However, extending our method to durative tasks is straightforward (see Subsection 7.3.2). Task announcement may be uncertain in both space and time. We write $P^A_\tau \in \text{Dist}(\mathbb{R}_{\geq 0})$ to denote the continuous distribution over the duration until announcement for task τ, and $P^L_\tau \in \text{Dist}(V)$ to denote the discrete location distribution for task τ, where $P^L_\tau(v)$ is the probability of $v \in V$ being the location task τ is announced at. In the next subsection, we compute these distributions over CTMC models of task announcement. For a task τ, we refer to the nodes $v \in V$ where $P^L_\tau(v) > 0$ as *potential task locations (PTLs)* for τ.

Each robot maintains a *schedule*, which is an ordered sequence of allocated tasks.

Definition 7.2. The *schedule* for robot r_i is a (possibly empty) sequence of tasks $\phi_i = \phi_i^1 \phi_i^2 ... \phi_i^k \in (T^*)$, where ϕ_i^j is the jth task τ executed by robot r_i, and $(T)^*$ denotes the set of sequences with elements of type T.

We write $|\phi_i|$ to denote the length of schedule ϕ_i. In Subsection 7.2.4, we consider schedules where robots only service a task if announced at a location they have been allocated.

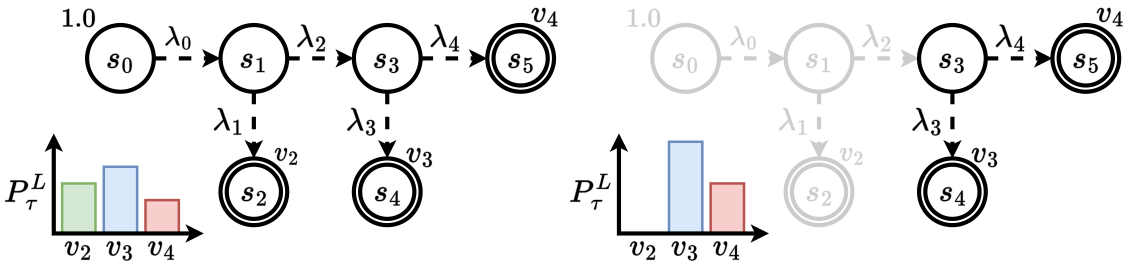
7.2.3 Task Announcement Model

In this subsection, we present *task CTMCs* as a model of stochastic task announcement in continuous time. Task CTMCs model the joint spatiotemporal announcement process, allowing for dependencies between the announcement time and location, such as in Example 7.1, where a longer announcement time infers the picker has travelled further.

Definition 7.3. A *task CTMC* for a task τ is a tuple $\mathcal{Q}_\tau = \langle S_\tau, init_\tau, \Delta_\tau, S^f_\tau, Lab_\tau \rangle$, where $S^f_\tau \subseteq S$ is a non-empty set of absorbing states, i.e. $E(s) = \Delta_\tau(s, s)$, or $E(s) = 0, \forall s \in S^f_\tau$, where reaching $s \in S^f_\tau$ represents the announcement of τ. $Lab_\tau : S^f_\tau \to V$ is a labelling function, where $Lab_\tau(s)$ is the physical announcement location for absorbing state $s \in S^f_\tau$.

Task CTMCs must have at least one absorbing state, where an absorbing state is eventually reached with probability 1, i.e. the task must eventually be announced. Task CTMCs are flexible and can model any nonnegative distribution to an arbitrary precision, as with PTDs. In fact, we can fit task CTMCs from empirical data using PTD techniques, such as the EM algorithm in [51] (see Appendix D).

For a task $\tau \in T$ represented as a task CTMC $\mathcal{Q}_\tau$, we can compute the announcement time and location distributions using standard model checking techniques [40, 48]. The expected announcement time $\mathbb{E}[P^A_\tau]$ is computed as the expected time to reach an absorbing state, as discussed in Subsection 2.3.3. The full announcement time distribution can be analysed through transient analysis [260] (see Subsection 2.3.1), but is expensive to compute exhaustively. To compute the discrete location distribution P^L_τ, we compute the reachability probabilities for each absorbing state as in Subsection 2.3.2, and sum them based on their labelling, i.e. their topological location. Formally:



(a) When the current state is s_0. **(b)** When the current state is s_3.

Figure 7.2: A task CTMC $\mathcal{Q}_\tau$ and the corresponding location distribution P_τ^L when at different states in the model. Concentric circles represent absorbing states, λ values are exponential rates, and the value above each absorbing state is the corresponding PTL, defined by the labelling function.

$$P_\tau^L(v) = \sum_{s \in \{s' \in S_\tau^f | Lab_\tau(s') = v\}} \Pr^{reach}(s). \tag{7.1}$$

Task CTMC state updates affect the task announcement time and location. We assume task CTMC states are fully observable. Observability is satisfied in many multi-robot problems, such as in Example 7.1, where human pickers are fit with GNSS trackers for robots to locate them [4]. For partially observable task CTMCs, we can compute the announcement distributions at each state by setting that state to be the initial state, and then using the techniques described above. The overall announcement distributions are then computed by weighting the distributions at each state with that state's probability in the current belief.

Example 7.2. Consider the task CTMC $\mathcal{Q}_\tau$ in Fig. 7.2, which models a fruit picker as per Example 7.1. The picker starts at s_0 (Fig. 7.2a) and will finish filling up their basket at location v_2, v_3, or v_4. If the picker reaches state s_3 (Fig. 7.2b), P_τ^A and P_τ^L reflect that the picker is closer to filling its basket, and that v_2 can no longer be the task location.

Task CTMCs support proactive decision-making, as robots can navigate towards PTLs using predictions of when and where tasks will be announced. However, the structure of a task CTMC affects the extent to which robots can exploit these predictions. We now introduce three possible task CTMC structures, which we use in Section 7.4 to assess the proactive MRTA framework we present in Section 7.3.

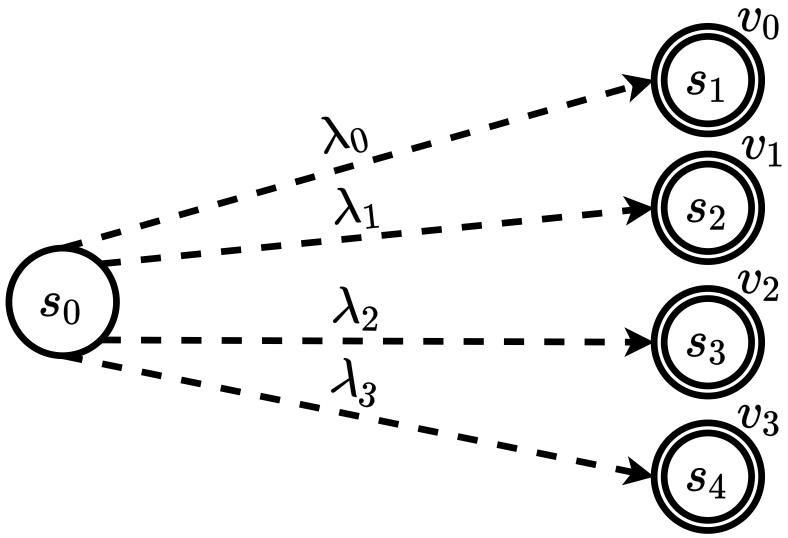


Figure 7.3: A task CTMC $\mathcal{Q}_\tau$ with a star topology, where concentric circles are absorbing states, λ values are exponential rates, and above each absorbing state is the corresponding PTL, as defined by the labelling function.

- *Contiguous Chain (CC)* CTMCs are structured as in Fig. 7.2, where the PTLs are contiguous on the topological map. This structure captures the behaviour of the fruit pickers in Example 7.1. The state of a CC CTMC is updated multiple times prior to announcement. State updates refine the announcement distributions as announcement gets nearer. This allows robots acting proactively to adjust their behaviour, which improves the chance of a robot being near the true task location upon announcement. Moreover, as the PTLs are contiguous, robots can navigate towards all PTLs simultaneously, which simplifies decision-making.

- *Discontiguous Chain (DC)* CTMCs are structured as in Fig. 7.2, where the PTLs are spread across the environment. Similar to CC CTMCs, robots can use DC CTMC state updates to modify their behaviour as task announcement draws closer. However, as the PTLs are discontiguous, decision-making becomes more difficult, as proactively navigating towards one PTL may move a robot further away from another.

- *Discontiguous Star (DS)* CTMCs are structured as in Fig. 7.3, where the PTLs are spread across the environment, similar to DC CTMCs. DS CTMCs

do not receive intermediate state updates, transitioning from the initial state directly to an absorbing state. With this, robots must make a single proactive decision using the initial announcement distributions, with no new information available until announcement.

A task $\tau \in T$ is announced at one of many PTLs $v \in V$, where each PTL-task pair (v, τ) can be viewed as a separate item to be allocated.

Definition 7.4. For a set of tasks T, the *set of PTL-task pairs* is written as:

$$PTL = \{(v, \tau) \mid v \in V \text{ and } \tau \in T \text{ and } P_\tau^L(v) > 0\}, \tag{7.2}$$

where (v, τ) is announced with probability $P_\tau^L(v)$ at a time distributed by P_τ^A.

In Section 7.3, we allocate each PTL-task pair individually. Let subscript i denote robot r_i, and superscript j denote the jth task τ in robot r_i's schedule. Robot r_i is allocated a subset of the PTLs for task τ, which we denote with V_i^j, where robot r_i is responsible for completing task τ if it is announced at a PTL in V_i^j.

7.2.4 MRTA Under Spatiotemporal Uncertainty

In this subsection, we formulate the MRTA problem tackled in this chapter, which corresponds to **Q5 - Task Allocation**. First, we redefine schedules to consider the PTLs each robot has been allocated for a task.

Definition 7.5. The *schedule* for robot r_i is written as $\phi_i = \phi_i^1 \phi_i^2 ... \phi_i^k \in (2^V \times T)^*$, where $\phi_i^j = (V_i^j, \tau)$, and $V_i^j \subseteq V$ is the set of PTLs for τ allocated to robot r_i.

For each $\phi_i^j = (V_i^j, \tau)$ and $v \in V_i^j$, $P_\tau^L(v) > 0$, i.e. all nodes in V_i^j are PTLs for task τ. Further, for all $\phi_i^j = (V_i^j, \tau)$ and $\phi_i^l = (V_i^l, \tau')$ where $j \neq l$, $\tau \neq \tau'$, i.e. a task may only appear once in ϕ_i. Next, we define allocations over the task set.

Definition 7.6. Let $R = \{r_1, .., r_n\}$ be a set of n robots and $T = \{\tau^1, ..., \tau^m\}$ be a set of tasks. An *allocation* is a mapping from robots to schedules $\Phi : R \to (2^V \times T)^*$ such that every PTL is allocated, i.e. $\bigcup_{i,j} \{(v, \tau) \mid v \in V_i^j \text{ and } (V_i^j, \tau) = \phi_i^j\} = PTL$.

In this chapter, we minimise *task waiting durations*, i.e. the duration between a task being announced and a robot beginning to service it at the task location. We refer to the time a robot begins servicing a task as the *service time*. Waiting durations are stochastic, due to uncertain task announcement and robot action durations.

Definition 7.7. Let W_τ^Φ be the waiting duration distribution for task $\tau \in T$ under allocation Φ. The *expected waiting duration*, $\mathbb{E}[W_\tau^\Phi]$, is given by:

$$\mathbb{E}[W_\tau^\Phi] = \mathbb{E}[B_\tau^\Phi - P_\tau^A] = \mathbb{E}[B_\tau^\Phi] - \mathbb{E}[P_\tau^A], \tag{7.3}$$

where B_τ^Φ is the service time distribution for task τ under allocation Φ, and P_τ^A is the announcement time distribution for task τ.

We describe how to evaluate Eq. 7.3 in Section 7.3. Finally, we define the MRTA problem under spatiotemporal uncertainty.

Problem 7.1. Given a team of n robots $R = \{r_1, ..., r_n\}$, and a set of m tasks $T = \{\tau^1, ..., \tau^m\}$, synthesise an allocation Φ^* that minimises the total expected waiting duration, i.e.:

$$\Phi^* = \underset{\Phi}{\operatorname{argmin}} \sum_{\tau \in T} \mathbb{E}[W_\tau^\Phi]. \tag{7.4}$$

7.3 Proactive SSI Auctioning Under Spatiotemporal Uncertainty

In this section, we present a framework that extends SSI for proactive task allocation using task CTMC models of stochastic task announcement to solve Problem 7.1. We auction each PTL-task pair $(v, \tau) \in PTL$ separately. In their schedules, robots group all PTLs they have been allocated for a task together, and consider them at the same time. We discuss this further in Subsection 7.3.2.

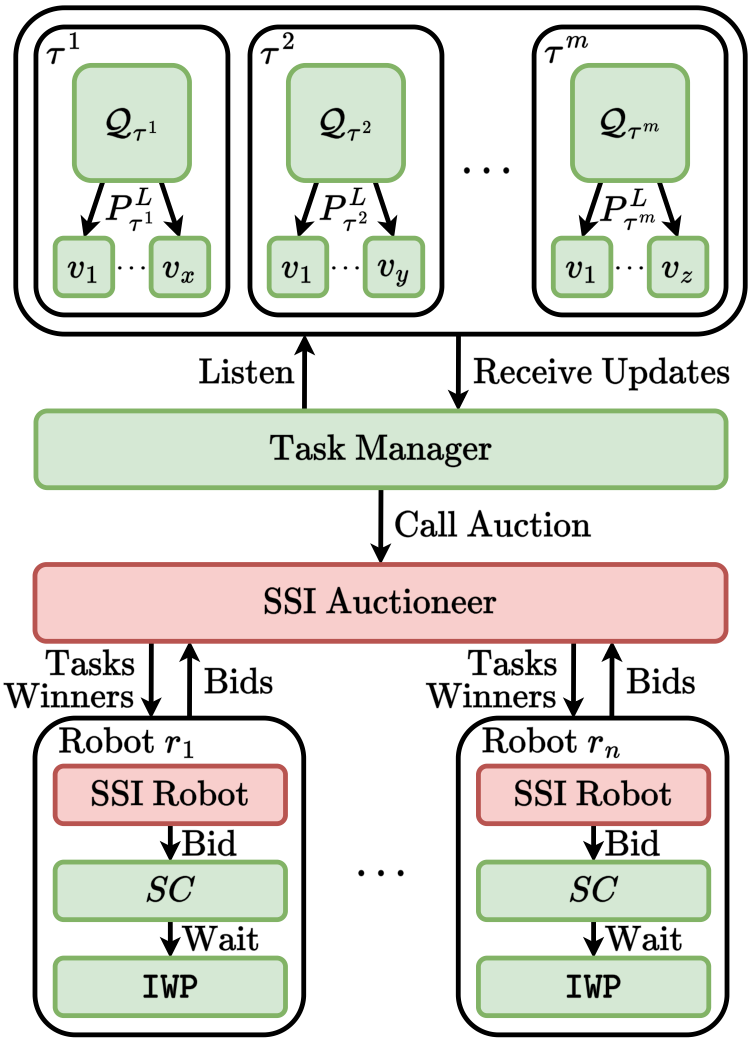


Figure 7.4: A framework that extends SSI for proactive task allocation. Red blocks are standard SSI; green blocks are our extensions.

7.3.1 Auctioning Framework

In Fig. 7.4, we present our proactive task allocation framework. The input to our framework is a set of task CTMCs $\mathcal{Q}_\tau, \forall \tau \in T$, which may be fit from empirical data, and the set of PTL-task pairs PTL. We then use a *task manager*, which listens for task CTMC state updates, and calls auctions for sets of PTL-task pairs. All auctions are conducted using SSI [200] (see Appendix E).

To synthesise an initial allocation, the task manager sends the full set of PTL-task pairs PTL to the auctioneer at the start of execution. Moreover, when the task manager receives task CTMC state updates during execution, it sends

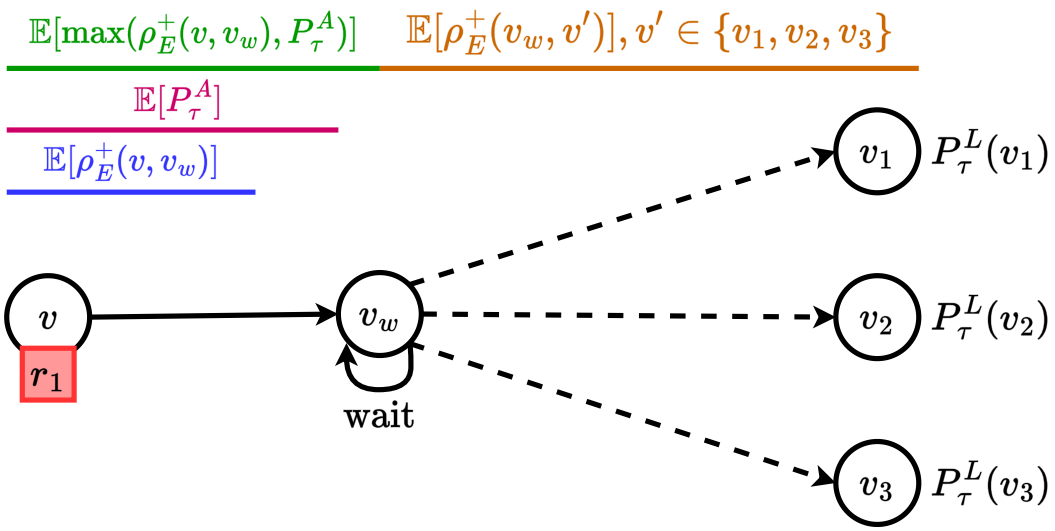


Figure 7.5: An example demonstrating how robots proactively execute tasks. Starting from node v at time 0, robot r_1 navigates to WP v_w and waits for task τ to be announced. Upon announcement, r_1 navigates to the task location and services τ, where the PTLs are v_1, v_2, and v_3. Arrows between different nodes represent multiple topological edges. The coloured values show the terms in Eq. 7.6 and the lower bound we use in practice.

the updated PTL-task pairs for that task to be reauctioned. Task CTMC state updates may change a PTL's probability; if this drops to 0, the PTL is not reauctioned, and removed from any schedules to stop robots from navigating to that location. Upon announcement, task $\tau \in T$ becomes deterministic, i.e. $P_\tau^L(v) = 1$ for the announcement location v, and $P_\tau^A = 0$; (v, τ) is then reauctioned for a final time. Online reauctioning provides robustness to delays in robot navigation and task announcement. For example, if a task at the end of a robot's schedule is announced early, reauctioning allows that robot, or another, to navigate to the task location immediately.

Next, we discuss our framework from the perspective of a single robot r_i, who begins by bidding for each PTL-task pair in the initial auction. The bid for PTL-task pair (v, τ) corresponds to the increase in r_i's schedule cost SC when inserting (v, τ) into schedule ϕ_i, where SC typically represents the schedule completion time or total distance travelled. In Subsection 7.3.2, we modify SC to compute the total expected waiting duration in robot r_i's schedule.

After the initial auction, robot r_i begins to proactively execute the first task τ in its schedule. To execute task τ proactively, r_i navigates to a *waiting point (WP)*

and waits there until τ is announced, as demonstrated in Fig. 7.5. Possible WPs include the location r_i begins to proactively execute τ, and the most likely PTL. In Subsection 7.3.3, we introduce *intermediate waiting points (IWPs)*, which are WPs that bring robots closer to the PTLs they have been allocated. IWPs minimise the expected waiting duration, and prevent costly backtracking manoeuvres which occur when a robot proactively navigates to the wrong location. We compare different WPs in Section 7.4, and use WPs in Subsection 7.3.2 to compute the cost of robot schedules. When task τ is announced, robot r_i observes the task location $v \in V$. If r_i was allocated v, it navigates there and services the task. Otherwise, it begins to navigate towards the WP for the next task in its schedule.

Upon a task CTMC state update, robot r_i bids on the updated PTL-task pairs for that task. If r_i loses the auction for a PTL-task pair it was assigned previously, the PTL is removed from r_i's schedule. If all PTLs for r_i's current task are removed, r_i moves onto its next task, whereas if at least one PTL is retained, r_i recomputes its current WP using the updated announcement distributions.

By auctioning each PTL-task pair separately, we allow multiple robots to be responsible for a single task, dependent on the announcement location. This allows for more flexible allocations, as if a task's PTLs are far apart, they can be allocated to different robots, reducing the waiting duration. Note that, upon announcement, tasks are serviced by a single robot.

Our proactive auction framework assumes that robots can communicate with a centralised auctioneer to receive tasks and submit bids. This differs from the congestion-aware planner in Chapter 6, where no robot communication is required to execute the synthesised policies. If robots can freely communicate during execution, they can periodically communicate their location to all other robots, which allows the context to be observed directly, rather than predicted through congestion distributions. However, this has a high communication overhead which imposes strong assumptions on the network. Communication with an auctioneer is possible on even intermittent networks, as communication is infrequent and the messages transmitted are small.

7.3.2 Computing Bids

In this subsection, we describe how robots compute bids that consider expected waiting durations. Robots group the PTLs they have been allocated for the jth task in their schedule τ into a set $V_i^j \subseteq V$, where robot r_i only services task τ if it is announced at a location in V_i^j. Concretely, when robot r_i inserts a PTL-task pair (v, τ) into its schedule ϕ_i during an auction, we find the current set of PTLs robot r_i has been allocated for task τ, V_i^j, where $(V_i^j, \tau) \in \phi_i$, and remove them; if there are no other PTLs, $V_i^j = \emptyset$. We then add v to V_i^j, $V_i^j = V_i^j \cup \{v\}$, and find the best insertion position for (V_i^j, τ) in ϕ_i. In Section 7.4, we use the cheapest insertion heuristic, which tests (V_i^j, τ) in each schedule position, and selects the position with minimal cost [200]. However, the cheapest insertion heuristic is suboptimal. The optimal insertion position can be found by solving a travelling salesperson problem, however this scales poorly with the schedule length.

Robots bid the increase in total expected waiting duration after adding the PTL-task pair (v, τ) into their schedule ϕ_i. This is captured by computing the schedule cost $SC(\phi_i)$ before and after (v, τ) is added, which computes the total expected waiting duration for schedule ϕ_i by modifying Eq. 7.4 to only consider tasks in ϕ_i.

Definition 7.8. Let $W_\tau^{\phi_i}$ and $B_\tau^{\phi_i}$ be the waiting duration and service time distributions for the PTLs robot r_i has been allocated for task τ. The *schedule cost* $SC(\phi_i)$ for robot r_i is defined as:

$$SC(\phi_i) = \sum_{(V_i^j, \tau) \in \phi_i} \mathbb{E}[W_\tau^{\phi_i}] = \sum_{(V_i^j, \tau) \in \phi_i} \mathbb{E}[B_\tau^{\phi_i}] - \sum_{(V_i^j, \tau) \in \phi_i} \mathbb{E}[P_\tau^A]. \tag{7.5}$$

The expected sum of announcement times in Eq. 7.5 is computed as in Subsection 7.2.3, and for the remainder of this subsection we describe how to compute the expected sum of service times, i.e. $\sum_{(V_i^j, \tau) \in \phi_i} \mathbb{E}[B_\tau^{\phi_i}]$. Recall that to execute tasks proactively, robots navigate to a WP and wait for task announcement. If the task is announced at a PTL the robot has been allocated, it navigates to the task location from the WP and services it, otherwise it navigates from the WP to its next task. Let $\rho_E^+(v, v')$ be the duration distribution for navigating along the shortest path between

v and v', i.e. $\sum_e \rho_E(e)$ for all $e \in E$ on the shortest path. To compute the expected sum of service times, we first define the expected service time for a task τ given the start time and location for the robot servicing τ, its WP, and the true task location.

Definition 7.9. Assume that task τ is announced at v', where τ is serviced by a robot who starts navigating towards WP v_w from node v at time t. The *service time* for τ is given by $B_\tau(v, t, v_w, v')$, where:

$$B_\tau(v, t, v_w, v') = \mathbb{E}[\max(t + \rho_E^+(v, v_w), P_\tau^A)] + \mathbb{E}[\rho_E^+(v_w, v')]. \tag{7.6}$$

We visualise Eq. 7.6 in Fig. 7.5. The first term in Eq. 7.6 represents the robot navigating to the WP and waiting for the task to be announced, and the second captures navigation from the WP to the task location. Further, by setting $v' = v_w$, Eq. 7.6 captures the time a robot leaves the WP for its next task if task τ is announced at a PTL the robot has not been allocated, as $\mathbb{E}[\rho_E^+(v_w, v')]$ becomes zero. In Eq. 7.6, we do not consider the duration since the last task CTMC state update, as due to the memoryless property of exponential distributions, $\mathbb{E}[P_\tau^A]$ remains constant until the next state update. In practice, the expectation over the maximum in Eq. 7.6 is expensive to compute. For example, if $\rho_E^+(v, v_w)$ and P_τ^A are represented as PTDs/CTMCs, the maximum is defined over the joint CTMC, which is impractical to model check exactly due to its size. Therefore, we use the lower bound $\max(\mathbb{E}[t + \rho_E^+(v, v_w)], \mathbb{E}[P_\tau^A])$ (see Fig. 7.5).

By assuming a known task location, the service time B_τ for task τ ignores location uncertainty, as distributed by P_τ^L. To compute the expected sum of service times, we must account for this uncertainty. Recall that we assume tasks are serviced instantaneously at the task location. Therefore, from the perspective of robot r_i the task is finished at the announcement location if it is a PTL they were allocated, and the WP otherwise. We capture this in a modified task location distribution $P_i^{L,j}$ for $\phi_i^j = (V_i^j, \tau)$, where the probability mass for the PTLs robot r_i has not been allocated, i.e. the PTLs not in V_i^j, is assigned to the WP, which

we denote as $v^j_{w,i} \in V$. Formally:

$$P_i^{L,j}(v) = \begin{cases} P_\tau^L(v) & \text{if } v \in V_i^j \text{ and } \phi_i^j = (V_i^j, \tau) \\ 1 - \sum_{v \in V_i^j} P_\tau^L(v) & \text{if } v = v^j_{w,i} \text{ and } \phi_i^j = (V_i^j, \tau) \\ 0 & \text{otherwise.} \end{cases} \tag{7.7}$$

In Eq. 7.8 we calculate the expected sum of service times for robot r_i, starting from its jth task τ. We do this recursively by adding the expected service time for τ to the expected sum of service times starting from its $j+1$th task τ'. However, the service time for the $j+1$th task τ' depends on the time and location r_i finishes its previous task τ and begins proactive execution for τ' (see Eq. 7.6). Robot r_i finishes its jth task τ at one of the PTLs it has been allocated or the WP, as defined by $P_i^{L,j}$, where each location has a different navigation duration from the WP $v^j_{w,i}$. Therefore, we compute the expected sum of service times for each $v' \in V$ robot r_i may finish its jth task τ at, and weight it by $P_i^{L,j}(v')$.

Definition 7.10. Assume robot r_i begins at node v at time t and executes the tasks in ϕ_i, starting from the jth task τ, i.e. $\phi_i^j = (V_i^j, \tau)$. The *expected sum of service times* is given by $\Omega_i^j(v,t)$, where:

$$\Omega_i^j(v,t) = \begin{cases} \sum_{v'} P_i^{L,j}(v') \cdot (t' + \Omega_i^{j+1}(v', t')) & \text{if } j \leq |\phi_i| \\ 0 & \text{if } j > |\phi_i|, \end{cases} \tag{7.8}$$

and $t' = B_\tau(v, t, v^j_{w,i}, v')$ is the expected service time for the jth task τ, given robot r_i starts at node v at time t, chooses WP $v^j_{w,i}$, and finishes task τ at v'.

If robot r_i starts at node v at time 0, $\Omega_i^1(v,0)$ corresponds to the expected sum of service times for schedule ϕ_i. With this, we rewrite the schedule cost $SC(\phi_i)$ in Eq. 7.5:

$$SC(\phi_i) = \Omega_i^1(v,0) - \sum_{(V_i^j, \tau) \in \phi_i} \mathbb{E}[P_\tau^A]. \tag{7.9}$$

If a tie occurs during bidding, we select the robot who reaches the WP for the task they are bidding on first. We can extend our framework to problems with durative tasks by modifying the recursive call in Eq. 7.8 to use the time and location the durative task is completed.

7.3.3 IWPs

In this subsection, we introduce IWPs, which are WPs that minimise the expected waiting duration for a task τ.

Definition 7.11. Let $\phi_i^j = (V_i^j, \tau)$ be the jth item in robot r_i's schedule. *IWPs* are defined according to $\texttt{IWP}_i^j : V \times \mathbb{R}_{\geq 0} \to V$, where $\texttt{IWP}_i^j(v, t)$ returns the IWP for the PTLs V_i^j robot r_i has been allocated for task τ, given r_i starts at node v at time t:

$$\texttt{IWP}_i^j(v, t) = \underset{v_w \in V}{\operatorname{argmin}} \sum_{v'} P_i^{L,j}(v') \cdot B_\tau(v, t, v_w, v'). \tag{7.10}$$

The IWP is a WP that minimises the expected service time B_τ for task τ given the PTLs robot r_i has been allocated. Minimising the expected service time minimises the expected waiting duration, as the expected announcement time (see Eq. 7.3) is independent of the WP. Further, when $v' = v_w$ in Eq. 7.10, the term in the summation captures r_i leaving v_w for its next task if τ is announced at a PTL r_i has not been allocated. If a tie occurs in Eq. 7.10, we select the WP v_w that minimises the expected duration from the WP to the task locations, i.e. $\sum_{v'} P_i^{L,j}(v') \cdot \mathbb{E}[\rho_E^+(v_w, v')]$. Eq. 7.10 is dependent on the time a robot starts a task, and so the IWP may change between bidding and execution, due to uncertain navigation durations and task announcement. Therefore, robots recompute IWPs online just before starting a task to improve decision-making.

Example 7.3. Consider Fig. 7.6, where robot r_1 is at v_0 at time 0, and has been allocated all PTLs for its jth task τ, where $P_\tau^L = \{v_1 : 0.4, v_2 : 0.3, v_4 : 0.3\}$, and $\mathbb{E}[P_\tau^A] = 7$. The IWP is $\texttt{IWP}_1^j(v_0, 0) = v_3$, where the weighted sum of B_τ for v_3 in Eq. 7.10 is $\max(6.1, 7) + 1.1 = 8.1$, compared to v_1, which evaluates to $0.4 \cdot (\max(5, 7) + 0) + 0.6 \cdot (\max(5, 7) + 2) = 8.2$.

In Eq. 7.10, we minimise over every possible WP in V, which is expensive for large environments. To reduce complexity, we use a greedy heuristic, where we remove nodes unlikely to be the IWP to compute a reduced set of candidate nodes to minimise over. Starting from the robot's initial location, we perform a breadth-first traversal, where neighbours are expanded only if their expected

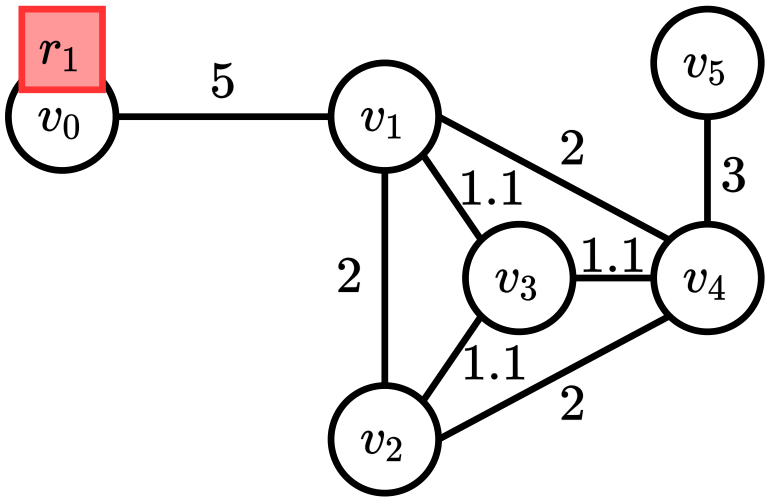


Figure 7.6: An example topological map, where edge weights are the expected duration of navigating that edge. Robot r_1 is initially located at node v_0.

duration to at least one task location is lower than the parent node. Moreover, if a neighbour has a lower duration for all task locations, we remove the parent from the candidate set, as the neighbour allows for more progress to the PTLs prior to announcement. This heuristic is suboptimal, and in some cases may remove the IWP from the candidate set, i.e. the minimal WP, as neighbours are greedily expanded towards the task locations. Despite this, our framework produces low waiting durations, as demonstrated in Section 7.4.

7.3.4 Complexity Analysis

In this subsection, we outline the time complexity of the algorithms run on the auctioneer and robots under our framework. We assume that the announcement time and announcement location distributions are precomputed, and ignore online updates, i.e. we just consider the initial auction.

Proactive Task Allocation (Auctioneer)

In each auction round, the auctioneer allocates a single PTL-task pair by determining the minimum bid. In the first round, there are $|PTL| \cdot n$ bids, one for each robot and PTL-task pair. The number of bids decreases by n each round until all PTL-task pairs are allocated. Minimisation is linear in the number of bids, and so the time

complexity for the auctioneer to run the initial auction corresponds to the total number of bids, given by $\sum_{k=1}^{|PTL|} k \cdot n = \mathcal{O}\Big(\frac{|PTL|^2+|PTL|}{2} \cdot n\Big)$, which is linear in the number of robots, and quadratic in the number of PTL-task pairs.

IWP Computation

During auctions, robots compute IWPs using Eq. 7.10, which finds the WP that minimises the waiting duration from a set of at most $|V|$ candidate nodes. We compute the expected service time for each candidate as a weighted sum over the PTLs. The time complexity for Eq. 7.10 is given by $\mathcal{O}(|V| \cdot |P^L_{max}|)$, where $|P^L_{max}|$ is the maximum number of PTLs for a single task.

Computing the Schedule Cost

Next, we consider the time complexity of calculating the schedule cost SC in Eq. 7.5, which corresponds to the complexity of the sum of service times in Eq. 7.8. Eq. 7.8 computes the IWP for the current task, given its start time and location, and makes multiple recursive calls for the next task, one for each of the current task's PTLs. The current task's announcement location affects the IWP for the next task. Therefore, we compute one IWP for the first task, $|P^L_{\max}|$ for the second, $|P^L_{\max}|^2$ for the third, and so on. Generally, for the jth task τ in ϕ_i, i.e. $\phi_i^j = (V_i^j, \tau)$, we compute $|P^L_{max}|^{j-1}$ IWPs. In total, we compute $\sum_{j=0}^{|\phi_i|-1} |P^L_{max}|^j$ IWPs, and so the time complexity is $\mathcal{O}\Big(\frac{|P^L_{max}|^{|\phi_i|}-1}{|P^L_{max}|-1} \cdot |V| \cdot |P^L_{max}|\Big)$, i.e. SC scales exponentially in the length of the schedule. In practice, we cache IWPs across multiple SC calls to improve scalability. We validate the scalability of our method for calculating SC empirically in Section 7.4.

Proactive Task Allocation (per Robot)

Finally, we consider the time complexity for a robot r_i to participate in a proactive auction, where we assume r_i uses the cheapest insertion heuristic to find the best schedule insertion position [200]. Robot r_i initially computes bids for each of the $|PTL|$ PTL-task pairs, and recomputes bids upon being allocated a PTL-task pair (see Appendix E). In the worst case, r_i always wins, and recomputes $|PTL| - (\sum_j |V_i^j|)$ bids for each of the $|PTL|$ rounds, where a new PTL-task pair is

added to ϕ_i each round. Each bid requires at most $|\phi_i|+1$ calls to SC, one for each schedule insertion position. Over the auction, robot r_i makes $\sum_{k=1}^{|PTL|}(|PTL|+1-k)\cdot k$ calls to SC. The time complexity of SC is dependent on $|\phi_i|$, which after k auction rounds is upper bounded by $\min(k,|T|)$, i.e. the number of PTL-task pairs that have been allocated, or the number of tasks, whichever is smaller. Therefore, the worst case time complexity is $\mathcal{O}\Big(\sum_{k=1}^{|PTL|}(|PTL|+1-k)\cdot k\cdot(\frac{|P^L_{max}|^{\min(k,|T|)}-1}{|P^L_{max}|-1}\cdot|V|\cdot|P^L_{max}|)\Big)$, where the exponential term $|P^L_{max}|^{\min(k,|T|)}$ from SC dominates, demonstrating that computing the schedule cost is a bottleneck in our framework. If one robot wins all PTL-task pairs, the other robots remain idle after computing their initial bids, i.e. their time complexity is significantly lower. However, the time complexity for auctioning all PTL-task pairs across the team is poor, as the schedule for the robot that always wins is increased in each round, which significantly increases the time required to recompute bids.

7.4 Experiments

In this section, we demonstrate the efficacy of our framework across multiple problems in simulation by comparing against a number of baselines. We simulate robot behaviour using CAMAS, as presented in Section 5.3. All experiments are run on Ubuntu 18.04, with an Intel Core i9-10900K CPU@3.7GHz and 32GB of RAM. All software is written in Python, except for PRISM [48], which we use for task CTMC analysis, which is written in Java/C++.

7.4.1 Experimental Setup

Experimental Environments

We consider two environments: the supermarket in Fig. 5.4, and a 10×10 grid world. Unlike the grid world, the aisles of the supermarket constrain movement, making it harder to find effective IWPs. Duration distributions for the grid world are synthetic, and fit using the moment matching techniques in [261] (see Appendix C). For the supermarket, we collect duration data in Gazebo using edge partitioning (see Section 4.4), where we simulate five Clearpath Jackals navigating using Move

Base Flex [247] for 24 hours. PTDs are fit from this data using the algorithm in [51] (see Appendix D).

Experimental Methods/Baselines

The core components of our framework are: proactive allocation; IWPs; and auctioning each PTL-task pair separately. We compare five methods, which use some or all of these components, in order to validate the design choices made in our framework:

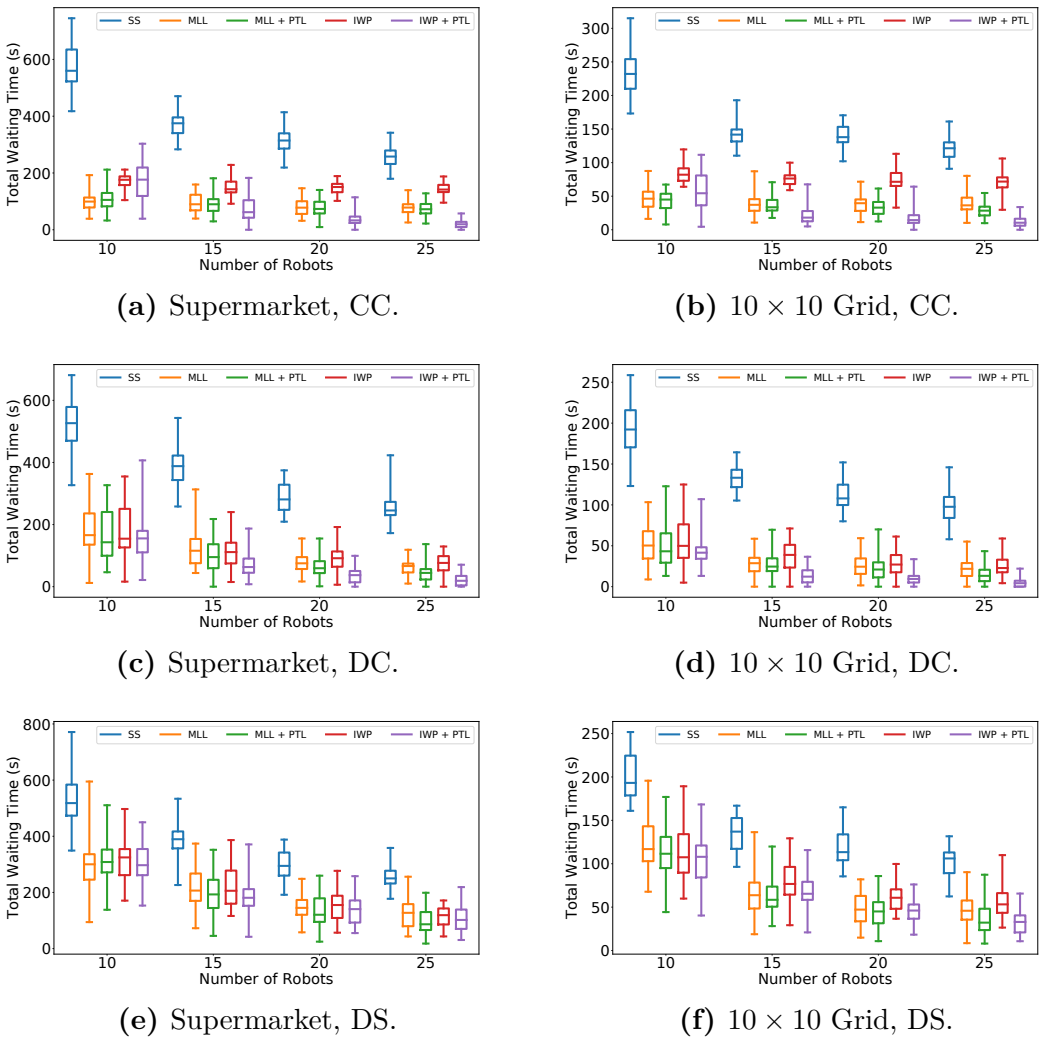
- *Stay Still (SS)*: A non-proactive MRTA technique, where each task is auctioned individually upon its announcement. As robots are not allocated tasks until they are announced, they wait at their initial location. This is representative of approaches which adapt SSI for online task announcement [200, 201, 205, 262].
- *Most Likely Location (MLL)*: During proactive execution, robots choose the most likely PTL as their WP for a task, instead of the IWP. During bidding, robots assume the MLL occurs with probability 1. PTLs are not auctioned separately, i.e. all PTLs for task $\tau \in T$ are assigned to a single robot who is responsible for servicing the task irrespective of its announcement location.
- *MLL+PTL*: The MLL method, but each PTL is auctioned separately. For each task, a robot's WP is the most likely PTL they have been allocated.
- *IWP*: Robots wait for announcement at the IWP for each task (see Subsection 7.3.3), but PTLs are not auctioned separately, as with MLL.
- *IWP+PTL*: Our complete framework, where robots wait at IWPs for task announcement, and each PTL is auctioned separately.

By comparing IWPs against the MLL, we aim to demonstrate the benefit of reasoning over the full task location distribution. Further, by comparing against methods which auction complete tasks rather than PTL-task pairs, we aim to show that allowing different robots to complete a task dependent on the announcement location reduces waiting durations. As discussed in Section 7.1, an alternative

approach to our MRTA framework is to plan on an extension of the MRMA (see Chapter 5) which captures the evolution of task CTMCs and allows for robot task selection. However, this approach cannot scale to realistic problems and so we do not compare against it here.

Experimental Problem Setup

In all problems, a team of robots must service a set of tasks whose announcement is modelled with task CTMCs. The initial robot locations are spread as evenly as possible across the environment (for the supermarket, this differs from the initial locations in Fig. 5.4), and tasks are serviced instantaneously upon a robot reaching the task location, where robots navigate using a shortest path planner. We would expect the congestion-aware planner in Chapter 6 to improve navigation performance, however we do not use it here as it reduces our ability to assess our proactive MRTA framework, as it becomes harder to identify whether performance is attributed to task allocation or planning. During bidding, robots use the cheapest insertion heuristic for schedule insertion [200]. We consider problems which use the task CTMC structures in Subsection 7.2.3, where CTMC transitions are sampled during execution, and all task CTMCs begin in their initial states at the start of execution. For each task CTMC type, we construct problems with 5-20 tasks in increments of 5, where the 10 task problem is the 5 task problem with 5 additional tasks etc. For CC CTMCs, we construct a single problem for each number of tasks which is static across repeats, where the expected announcement times for each task are 60 seconds apart. CC CTMCs are as in Fig. 7.2, i.e. there are three PTLs, where the earliest PTL has probability 0.4, and the other two have probability 0.3. For DC and DS CTMCs, we construct a different problem for each repeat for a given number of tasks. For each problem, the DC CTMCs have three PTLs, as in Fig. 7.2, and the DS CTMCs have four PTLs, as in Fig. 7.3. The expected announcement times, PTLs, and announcement location distributions for each problem are randomised, but kept consistent between methods.



(a) Supermarket, CC. **(b)** 10 × 10 Grid, CC.

(c) Supermarket, DC. **(d)** 10 × 10 Grid, DC.

(e) Supermarket, DS. **(f)** 10 × 10 Grid, DS.

Figure 7.7: The performance of each method as we increase the team size.

7.4.2 Measuring Performance as Robots Increase

First, we analyse performance as we increase the team size. In each environment, we consider the 20 task problem for each task CTMC type, and run 40 repeats to measure the total waiting duration for 10-25 robot teams, in increments of 5. In Fig. 7.7 we present the observed waiting durations, and in Table 7.1 we show the mean waiting durations alongside the results of a one-sided Mann-Whitney U statistical significance test, where $p = 0.05$. In 12/24 of the tested problem configurations, IWP+PTL produces significantly lower waiting durations than all other methods. In 10 problems, IWP+PTL ties with other methods, i.e. no

Table 7.1: The mean waiting durations for the results in Fig. 7.7. The problem column is written <environment>-<CTMC structure>-<number of robots>, where Sup denotes the supermarket, and Grid denotes the 10×10 grid world. Bold values have the lowest mean for that problem. Methods with highlighted cells do not produce statistically significantly higher waiting durations than the method with lowest mean, according to a one-sided Mann-Whitney U test with $p = 0.05$.

Problem	SS	MLL	MLL+PTL	IWP	IWP+PTL
Sup-CC-10	576.39	**101.51**	106.36	170.83	172.51
Sup-CC-15	373.24	93.31	88.41	148.99	**71.43**
Sup-CC-20	316.21	80.19	75.83	146.26	**36.80**
Sup-CC-25	257.82	76.70	73.19	144.47	**20.87**
Sup-DC-10	522.58	179.21	163.17	172.97	**152.59**
Sup-DC-15	379.45	123.29	100.56	113.85	**71.06**
Sup-DC-20	286.28	76.58	62.27	92.09	**37.39**
Sup-DC-25	252.59	61.64	43.03	72.50	**22.56**
Sup-DS-10	532.91	**296.74**	311.00	310.03	305.23
Sup-DS-15	388.00	211.94	193.77	221.02	**190.41**
Sup-DS-20	295.21	148.08	133.36	157.77	**132.85**
Sup-DS-25	254.63	124.24	**97.42**	114.22	104.80
Grid-CC-10	233.23	47.86	**42.72**	83.57	58.04
Grid-CC-15	141.89	38.72	37.47	76.10	**22.86**
Grid-CC-20	140.54	38.64	33.21	73.54	**16.57**
Grid-CC-25	120.32	38.21	28.23	70.94	**11.35**
Grid-DC-10	191.16	52.20	49.06	55.48	**43.17**
Grid-DC-15	132.86	28.08	27.42	37.29	**13.93**
Grid-DC-20	111.57	26.51	22.68	28.38	**10.51**
Grid-DC-25	99.68	22.33	14.08	25.04	**5.15**
Grid-DS-10	199.34	119.40	110.30	113.95	**105.60**
Grid-DS-15	134.35	64.15	**63.21**	79.20	68.47
Grid-DS-20	118.84	49.14	**44.62**	62.75	45.37
Grid-DS-25	102.51	47.98	35.59	55.09	**32.34**

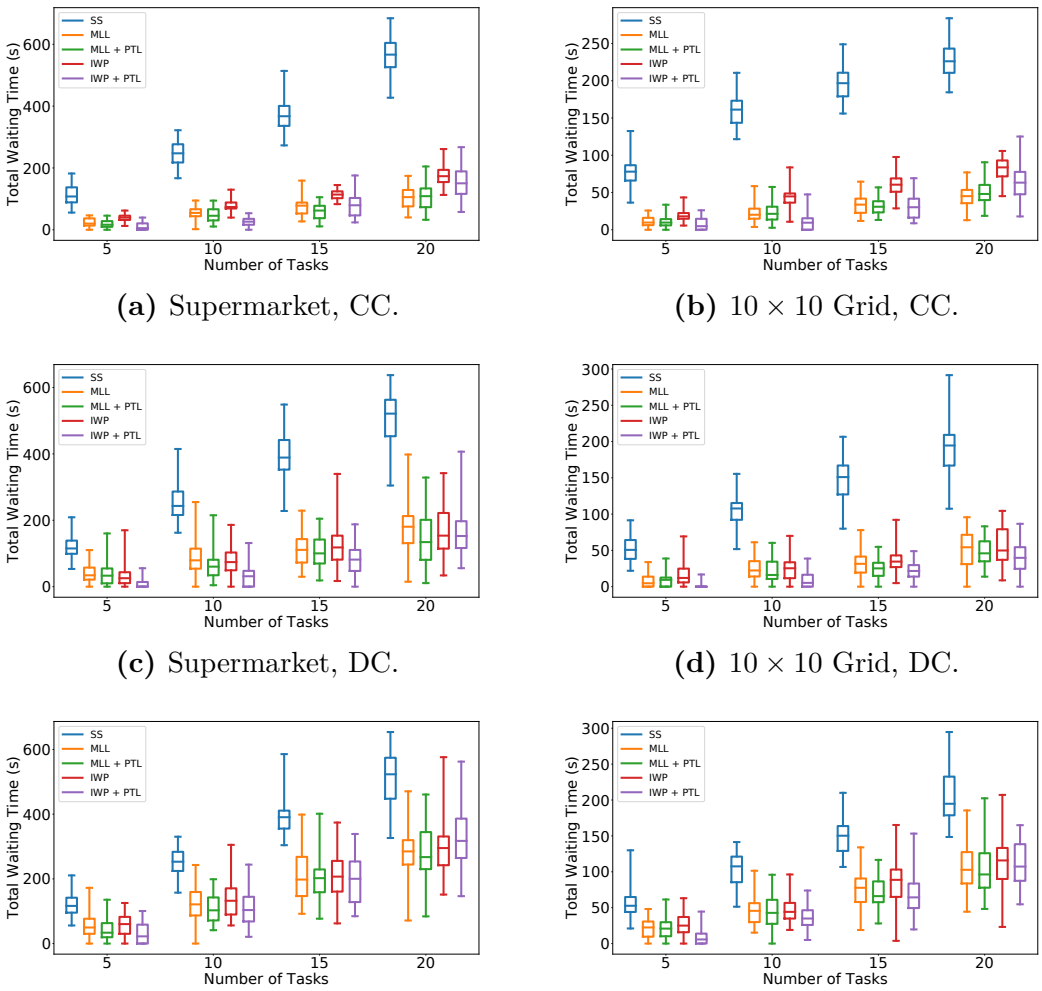
method is significantly better than IWP+PTL, but some are statistically similar. In the smallest CC problems in both environments, the MLL and MLL+PTL waiting durations are significantly less than IWP+PTL. In all problems, the SS waiting durations are significantly higher than every other method. As tasks are not allocated until announcement in SS, robots remain idle at their initial location, which poorly utilises the MRS.

For larger teams, there is more likely to be a robot near the PTLs, decreasing the waiting duration. Further, the gap between methods which auction PTLs

separately and those that don't increases with the team size, as PTLs are split between robots to minimise the waiting duration. Auctioning PTLs separately utilises robot redundancy, such as when we have more robots than tasks. Moreover, it extends the notion of spatial dependencies in SSI [200], as if PTLs for different tasks are close to each other, they are likely to be allocated to the same robot.

IWP+PTL decreases waiting durations by reasoning over the full task location distribution, and splitting PTLs between robots. However, in many problems the related IWP method performs worse than simpler MLL-based methods. In the CC problems, IWP sends robots to the middle of the three contiguous PTLs, as unlike the MLL method it reasons over the full location distribution. In the CC task CTMCs, we discover if the MLL is the announcement location in the second state update. With this, the MLL method will either incur a waiting duration of zero, or have time to move to one of the remaining PTLs. For IWP, the robots have to move if the MLL is the announcement location, increasing the waiting duration. This could be avoided by modelling task CTMC state updates during bidding, however this would increase complexity. Moreover, if the PTLs for a task are far apart, robots will still have to travel far after announcement using the IWP method. IWP+PTL avoids this by sending different robots to different PTLs. As the team size decreases, IWP+PTL assigns more PTLs to a single-robot, bringing performance closer to the IWP method.

The MLL-based methods perform competitively in many problems, and are less computationally intensive, as we assume the MLL is the announcement location. However, robots risk backtracking if the MLL is not the announcement location, which increases the waiting duration. In the CC problems, backtracking never occurs, as the PTLs are contiguous, and so MLL and MLL+PTL perform well. The MLL+PTL method typically performs worse than IWP+PTL. If a robot using MLL+PTL has been assigned the MLL, all other PTLs for that task can be added to the schedule with zero cost, as robots assume the MLL is the announcement location. Therefore, the MLL+PTL robot will be assigned all PTLs for a task,



(a) Supermarket, CC. **(b)** 10×10 Grid, CC.

(c) Supermarket, DC. **(d)** 10×10 Grid, DC.

(e) Supermarket, DS. **(f)** 10×10 Grid, DS.

Figure 7.8: The performance of each method as we increase the number of tasks.

causing behaviour similar to MLL. IWP+PTL avoids this by reasoning over the full location distribution.

In the DS problems, the proactive methods perform similarly, i.e. there is always a tie for the best method. This is likely due to a lack of structure in the task CTMCs. In the CC problems, task locations are contiguous, and the location distribution is updated over time, which allows for allocations to be improved online. Though the DC problems lose the spatial structure, the location distribution still receives updates, which we can exploit. Conversely, the DS problems are one-shot problems, i.e. there are no updates, and no relationship between the task locations, which makes it harder to proactively coordinate the robots.

Table 7.2: The mean waiting durations for the results in Fig. 7.8. The table is structured as in Table 7.1, except for the number in the problem column, which now refers to the number of tasks.

Problem	SS	MLL	MLL+PTL	IWP	IWP+PTL
Sup-CC-5	110.18	23.49	18.28	38.98	**10.86**
Sup-CC-10	245.73	54.37	47.51	78.30	**25.01**
Sup-CC-15	373.90	73.97	**57.87**	113.25	79.70
Sup-CC-20	564.27	**103.24**	106.00	177.85	153.98
Sup-DC-5	121.30	40.14	36.96	38.39	**8.59**
Sup-DC-10	251.18	87.79	61.18	77.77	**33.29**
Sup-DC-15	393.72	113.51	105.25	127.55	**84.52**
Sup-DC-20	509.11	178.26	**143.80**	167.04	167.91
Sup-DS-5	118.80	55.50	44.10	57.47	**32.05**
Sup-DS-10	250.17	127.77	**108.28**	138.49	114.16
Sup-DS-15	388.79	211.16	**199.19**	212.87	199.56
Sup-DS-20	513.09	286.47	**276.51**	306.26	325.05
Grid-CC-5	79.85	11.28	11.40	19.02	**7.65**
Grid-CC-10	160.36	22.33	22.35	43.62	**11.96**
Grid-CC-15	196.97	34.93	**32.09**	61.40	32.42
Grid-CC-20	228.05	**45.52**	49.30	82.39	62.37
Grid-DC-5	52.91	7.75	8.74	16.58	**1.96**
Grid-DC-10	103.80	24.18	21.65	26.63	**9.41**
Grid-DC-15	146.13	32.92	24.85	35.81	**22.11**
Grid-DC-20	189.62	52.31	47.19	55.13	**38.68**
Grid-DS-5	55.78	21.80	22.02	26.97	**8.44**
Grid-DS-10	105.44	45.91	46.22	46.54	**36.29**
Grid-DS-15	149.20	75.45	71.32	83.42	**69.19**
Grid-DS-20	205.88	**104.25**	106.74	113.76	110.91

7.4.3 Measuring Performance as Tasks Increase

Next, we consider how the number of tasks affects performance. For a 10 robot team, we consider the 5-20 task problems for each task CTMC type, and run 40 repeats to measure the total waiting duration. The observed waiting durations are shown in Fig. 7.8, and the mean waiting durations with statistical significance results are presented in Table 7.2, as in Table 7.1.

IWP+PTL produces significantly lower waiting durations in 12/24 problems, ties in 8/24, and is beaten in 4/24. Generally, the trends observed when increasing the tasks corroborate with what we saw in the previous experiment. IWP+PTL

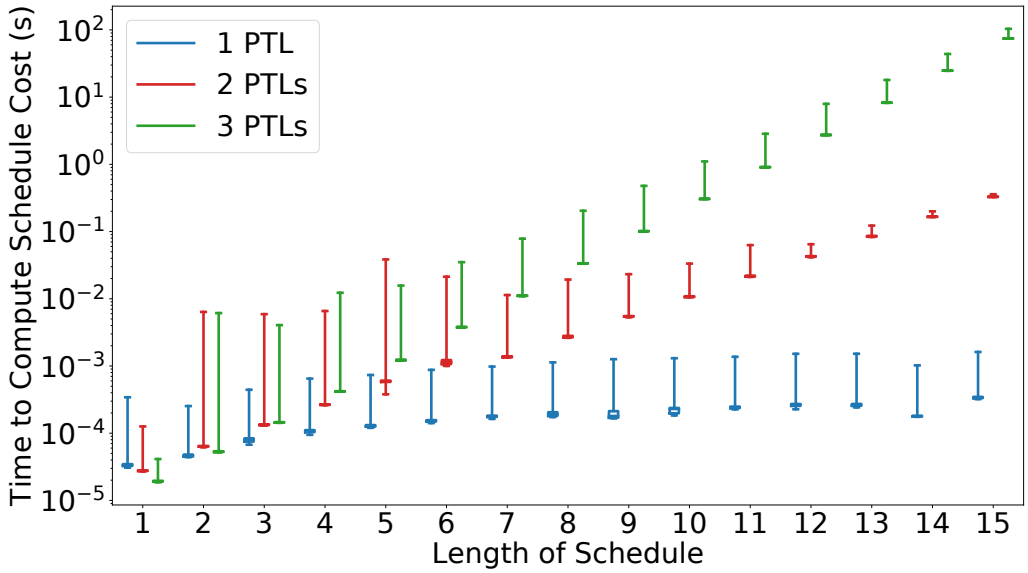


Figure 7.9: Scalability of the schedule cost SC.

performance decreases as the tasks increase, as there is less redundancy in the MRS, and less opportunity to distribute PTLs across the robots. Moreover, the MLL-based methods typically outperform the IWP-based methods on CC problems with larger task sets, as the online state updates allow MLL-based methods time to react if the MLL is not the task location, and to serve tasks with zero waiting duration if it is.

7.4.4 Schedule Cost Scalability

In Fig. 7.9, we demonstrate the exponential scalability of computing the schedule cost SC, as discussed in Subsection 7.3.4. Computing the cost of robot schedules is a bottleneck in our framework. We run 40 repeats of SC for schedule lengths ranging from 1-15, and for tasks with 1-3 PTLs, where schedules contain all PTLs for a task. These results also act as a proxy for comparing the scalability of IWP and MLL-based methods, where the core difference is in the schedule cost function. IWP-based methods vary in scalability dependent on the PTLs per task, whereas MLL-based methods consider only one PTL, the MLL. This significantly improves scalability, but at the cost of higher waiting durations, as discussed earlier in this section.

7.5 Discussion

In the previous section, we demonstrated that proactive task allocation decreases waiting durations. We compared against a non-proactive baseline which adapts SSI to online settings. SSI is suboptimal, and is at most a factor of two from the optimal for sum of cost objectives [200]. The performance of this baseline could be improved by using an optimal MRTA method such as combinatorial auctions [199]. However, this is still unlikely to outperform proactive methods. For example, the SS waiting durations in Section 7.4 are often more than two times higher than the best proactive method. Moreover, our proactive framework can be adapted to use less suboptimal MRTA methods in a straightforward way. To effectively allocate tasks proactively we require accurate announcement models, which can be constructed from empirical data, similar to navigation duration distributions (see Section 4.4). If announcement data is not available, non-proactive methods may be a suitable alternative.

Across both experiments in Section 7.4, IWP+PTL is significantly better in 50% of problems, and is unbeaten in 87.5%. However, the IWP-based methods are less scalable, as they reason over the full task location distribution during bidding (see Subsection 7.4.4). The MLL-based methods are much faster, as schedule evaluation scales linearly in the number of tasks rather than exponentially. Further, in 50% of the problems we evaluated, at least one of the MLL baselines is statistically similar to the method with lowest waiting durations. Therefore, if there are many tasks, or if computation time is limited, MLL-based methods can be used to effectively allocate tasks proactively. Moreover, MLL-based methods perform well when the PTLs are close together, as navigating towards the MLL brings the robot nearer to all PTLs, reducing waiting durations.

The performance of our framework is sensitive to the structure embedded in the task CTMCs, which we exploit during allocation. If there is spatial structure, e.g. the PTLs are close together, then it is easier to find effective IWPs. If there is temporal structure, i.e. there are multiple task CTMC state updates before announcement, then we can improve our allocations online. This structure exists in many real-world problems. For example, in the agricultural problem in Example 7.1,

there is a task CTMC state update each time the picker reaches a new node, which rules out the previous node as a PTL. Alternatively, for order picking tasks in a warehouse, commonly ordered items are often placed near packing stations, i.e. the most likely PTLs are often clustered within a small area [263]. To attain the best performance, this structure should be captured while constructing the task CTMCs. If there is no inherent structure in the task announcement process, the proactive methods in Section 7.4 perform similarly, and so alternative criteria should be used to select the most appropriate method, such as scalability.

7.6 Summary

For many problems, such as robots assisting fruit pickers, the announcement of robot tasks is stochastic in both space and time. To address **Q5 - Task Allocation** and efficiently allocate tasks with uncertain announcements, we must explicitly model the spatiotemporal processes which govern task announcement. In this chapter, we model stochastic task announcement in continuous time using CTMCs, and present a framework which extends SSI for proactive task allocation, i.e. robots navigate towards task locations before announcement. Each PTL is auctioned separately, which allows different robots to be responsible for a task dependent on the announcement location. To handle uncertain task locations, robots wait at IWPs which minimise the expected waiting duration. This prevents robots from committing to a single PTL too early, which can lead to costly backtracking manoeuvres if the task is announced at a different location. Our framework allows for coordination under global objectives, overcoming a limitation of the congestion-aware planner, and can solve any MRTA problem where task announcement is stochastic, provided that all tasks are known and eventually announced. Further, our method is unbeaten in 87.5% of the problems we evaluated, where we compared against methods which do not allocate tasks proactively, as is commonly seen in the literature, and methods which do not use IWPs or do not auction PTLs separately. By modelling stochastic task announcement, we capture sources of temporal uncertainty which affect robot execution but do not directly affect action

durations, which contributes towards **Q2 - Modelling**. Our proactive MRTA framework has previously appeared in [27].

8 Conclusion

Contents

8.1 Summary of Contributions 171
8.2 Future Work 175

8.1 Summary of Contributions

In this thesis, we addressed multi-robot coordination under temporal uncertainty using modelling, planning, policy evaluation, and task allocation techniques. Our solutions capture temporal uncertainty explicitly during coordination by modelling stochastic action durations, the effects of robot interactions on navigation, and stochastic task announcement. With this, we synthesise behaviour that is robust to temporal disturbances during execution, as our expectations of behaviour during planning closely approximates what we observe during execution. We now summarise our contributions to each of the five research questions posed in Chapter 1.

Q1 - Interactions: How do robot interactions affect action durations?

In Chapter 4 we introduced congestion, a mode of interaction for mobile robots which occurs when multiple robots share the same physical space. We measure

congestion in terms of robot presence at map resources when a new robot begins to navigate. We observed that robot navigation durations increase with congestion, due to the increased difficulty of avoidance manoeuvres. Congestion also increases duration uncertainty, as there is increased variation over how congestion occurs. Further, we observed spatial dependencies between map resources, as robots on one map resource may affect a robot navigating on another, which we capture with map resource groups. We also introduced the notion of contexts for representing phenomena which affect robot action durations, where we use scalar and vector contexts to describe congestion. From observing the effects of congestion, we found that in some instances similar contexts show no statistical difference in duration effects, and introduced interaction bands to group these contexts together.

Q2 - Modelling: Can we build probabilistic models that accurately capture temporal uncertainty and robot interactions?

In Chapter 4 we constructed probabilistic duration models which capture congestion and other phenomena that affect action durations. Duration models are represented with PTDs fit from empirical data, where data is collected using random navigation or edge partitioning (see Section 4.4). With this, in Chapter 5, we presented an MRMA which models asynchronous, continuous-time multi-robot execution. The MRMA detects robot interactions in the joint state at the instant an action is triggered. Though there exist other continuous-time models for asynchronous MRSs [19, 20, 22, 52], the MRMA is the first to explicitly model execution-time robot interactions and their effects on action durations. Empirically, we demonstrated that the MRMA accurately captures the high-level behaviour of MRSs simulated in Gazebo. To avoid the complexity of MRMA planning, in Chapter 6 we introduced TVMAs as a model for single-robot planning under congestion. Our TVMAs capture the effects of congestion with time-varying congestion distributions at each topological edge. Further, we presented route CTMCs as a model for continuous-time policy execution under congestion, which support the computation of congestion distributions. Though our congestion

distributions are approximate, they allow us to effectively reason over congestion, as demonstrated empirically in Section 6.8. To synthesise robust behaviour, we must also model sources of temporal uncertainty which do not directly affect action durations. In Chapter 7 we presented task CTMCs as a model of stochastic task announcement, which affect the time robots can service a task.

Q3 - Planning: Can we use these models to construct effective planning mechanisms?

In Chapter 6, we presented a framework for multi-robot planning under congestion. Robots plan sequentially, where each robot considers those who planned before. At the core of our framework is a PRT, which stores route CTMCs capturing robot policy execution, and computes distributions over congestion. We build a TVMA for each robot, and synthesise a time-dependent policy using LRTDP-TVMA, which incrementally builds and solves an MDP abstraction of the robot's TVMA. By synthesising time-dependent policies on congestion-aware models, we produce robot behaviour that is robust to temporal disturbances caused by congestion and other unmodelled sources of temporal uncertainty. Our framework outperforms methods which ignore congestion during planning, or try to avoid congestion entirely.

Q4 - Policy Evaluation: Can we apply policy evaluation techniques on these models to evaluate task-level properties of execution-time multi-robot behaviour?

In Section 5.3 we presented CAMAS, which evaluates task-level metrics of multi-robot performance, such as the time to complete a set of tasks, by sampling through the MRMA given a set of robot policies. CAMAS simulates behaviour significantly faster than real-time, and can evaluate the behaviour of an MRS simulated in Gazebo (see Section 5.5). We use sampling-based techniques as it is intractable to apply exact model checking techniques to the MRMA due to its size. To the best of our knowledge, CAMAS is the first DES to explicitly model the effects of robot interactions on action durations. Alternatively, in Section 6.7, we applied exact model checking techniques to evaluate the smaller single-robot route

CTMCs in the PRT, which describe the continuous-time behaviour induced by the policies synthesised by the congestion-aware planner. Route CTMCs approximate the effects of the other robots with congestion distributions. However, the models in the PRT after planning are inaccurate, due to the sequential planning assumption (see Chapter 6). To improve this, we presented an iterative procedure which refines route CTMCs to more accurately capture congestion. The refined CTMCs closely approximate what we observe during execution, and can accurately evaluate timed single-robot properties (see Section 6.8).

Q5 - Task Allocation: Can we allocate tasks efficiently under spatiotemporal uncertainty?

In Chapter 7, we presented a proactive MRTA framework for tasks whose announcement is uncertain in space and time. Our framework represents stochastic announcement with task CTMCs, and auctions each PTL separately using SSI. Robots navigate to task locations proactively using IWPs which balance between the PTLs. Our framework synthesises allocations which produce low waiting times, i.e. robots reach the task location shortly after announcement. In particular, we outperform methods which do not use IWPs, auction PTLs separately, or allocate tasks proactively, as demonstrated in Section 7.4.

Solution methods in this thesis have been utilised by other researchers. The CAMAS simulator (see Section 5.3) has been used by Accenture Labs to optimise warehouse configurations for MRSs. CAMAS rapidly evaluates different configurations, allowing logistics companies to analyse how a given warehouse layout affects multi-robot performance before committing to it. Further, the congestion-aware planner in Chapter 6 has been used by the University of Lincoln to coordinate a fleet of agricultural robots operating alongside human fruit pickers. The congestion-aware planner effectively distributes robots across narrow fruit fields, where any congestion significantly affects navigation performance.

8.2 Future Work

The work in this thesis opens up multiple avenues for future research. To fit accurate probabilistic duration models, we require large navigation duration datasets (see Section 4.4). Data collection is cumbersome for large environments, or those with little regularity. Moreover, we require data for each environment we deploy our MRS in. In future work, we could investigate how duration distributions can be generalised across different map resources and environments. One approach could be to extend edge partitioning (see Section 4.4) to collect data for map structures larger than single edges, such as T junctions, and share this data across all occurrences of that structure, possibly in different environments.

The MRMA in Chapter 5 accurately captures multi-robot execution, where CAMAS simulates behaviour faster than real-time. In the future, we could use the MRMA as a model-based surrogate for reinforcement learning, where episodes are sampled using CAMAS. Training on the MRMA has the potential to speed up learning, without sacrificing the quality of the learned policy. Policies learned on CAMAS could then be transferred to real robots, which avoids training in the real world or in physics-based simulators such as Gazebo, where episodes are run close to real-time. Moreover, we could extend CAMAS to use statistical model checking techniques for MA [177], expanding the properties we can evaluate.

A core extension of our work is integrating the congestion-aware planner in Chapter 6 with the proactive MRTA framework in Chapter 7. However, the congestion-aware framework is designed for offline planning, whereas in Chapter 7, robots replan online dependent on task announcement. We could adapt the congestion-aware planner for online settings with two PRT modifications. The PRT can be accessed by only one robot at a time. For offline problems, this is handled by the priority ordering. Online, the priority ordering must be replaced with an access rule, such as first come first served, for robots to query the PRT and insert route CTMCs. Moreover, robots could start tasks at different times. To handle this, the PRT must log the start time of each robot's route. When

computing congestion probabilities, the probability of a robot being on an edge before their start time is zero.

To improve the congestion-aware planner, we could consider more complex tasks which require explicit robot collaboration, such as handover tasks. This poses difficulty for the sequential planning assumption, as collaborating robots require knowledge of each other during planning. One approach is to identify the collaborating robots and model them jointly, similar to the MRMA. Though these joint models would be smaller than the full MRMA, they would be too large for planning. An alternative is to use techniques for planning under sparse interactions [101, 112] (see Subsection 3.1.5), where robots only act jointly when collaboration is required.

A further congestion-aware planner extension could be to adapt LRTDP-TVMA to discretise the PTDs, rather than just use the expected duration. For example, we could approximate PTDs as a discrete distribution using the technique in [264], and branch by this distribution in the MDP abstraction of the TVMA. This could improve the synthesised policies and the time to reach the goal, as the temporal dynamics of robot actions are more accurately captured in the MDP, but would also increase the MDP branching factor and thus increase the complexity of planning.

In the proactive MRTA framework in Chapter 7, the schedule cost function scales exponentially in the schedule length (see Subsection 7.3.4). Currently, we allocate all PTLs at the start of execution, however some tasks won't be announced for a significant time. With this, we could consider batching PTLs based on their expected announcement time, and auctioning batches with earlier announcements first. By staggering allocation, we reduce the length of robot schedules, improving scalability. Task batching could also be used to prioritise high probability PTLs.

The auctioneer in Chapter 7 has a global view of the MRS not possessed by the robots. We could use this to synthesise allocations that provide guaranteed multi-robot behaviour. For example, we may want to guarantee that all tasks are serviced within two minutes of announcement. For this, we could investigate MRTA solutions which satisfy temporal logic formulae [185], and approaches for optimising conditional value at risk [265].

References

[1] Luca Iocchi, Daniele Nardi, and Massimiliano Salerno. "Reactivity and Deliberation: A Survey on Multi-Robot Systems". In: *Proceedings of the Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems*. Springer. 2000, pp. 9–32.

[2] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. "A Survey and Analysis of Multi-Robot Coordination". In: *International Journal of Advanced Robotic Systems* 10.12 (2013), p. 399.

[3] Ocado Group. *What is an Ocado CFC?* 2021. URL: `https://www.ocadogroup.com/about-us/what-we-do/automated-ocado-customer-fulfilment-centre` (visited on 03/28/2022).

[4] Muhammad W Khan, Gautham P Das, Marc Hanheide, and Grzegorz Cielniak. "Incorporating Spatial Constraints into a Bayesian Tracking Framework for Improved Localisation in Agricultural Environments". In: *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 2440–2445.

[5] Robotics 24/7 Staff. *AutoX Passes 1000 Vehicle Milestone for Its RoboTaxi Fleet, the Largest in China*. 2022. URL: `https://www.robotics247.com/article/autox_passes_1000_robotaxi_fleet_milestone_expands_san_francisco_testing` (visited on 03/28/2022).

[6] Gautham Das, Grzegorz Cielniak, Pal From, and Marc Hanheide. "Discrete Event Simulations for Scalability Analysis of Robotic In-Field Logistics in Agriculture–A Case Study". In: *Proceedings of the IEEE International Conference on Robotics and Automation, Workshop on Robotic Vision and Action in Agriculture (WRVAA)*. 2018.

[7] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. "A Comprehensive Taxonomy for Multi-Robot Task Allocation". In: *The International Journal of Robotics Research* 32.12 (2013), pp. 1495–1512.

[8] Ola Pettersson. "Execution Monitoring in Robotics: A Survey". In: *Robotics and Autonomous Systems* 53.2 (2005), pp. 73–88.

[9] Bruce Golden, Arjang Assad, Larry Levy, and Filip Gheysens. "The Fleet Size and Mix Vehicle Routing Problem". In: *Computers & Operations Research* 11.1 (1984), pp. 49–66.

[10] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[11] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and Acting in Partially Observable Stochastic Domains". In: *Artificial Intelligence* 101.1-2 (1998), pp. 99–134.

[12] Hang Ma, TK Satish Kumar, and Sven Koenig. "Multi-Agent Path Finding with Delay Probabilities". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. 2017, pp. 3605–3612.

[13] Milan Tomy, Bruno Lacerda, Nick Hawes, and Jeremy L Wyatt. "Battery Charge Scheduling in Long-Life Autonomous Mobile Robots via Multi-Objective Decision Making Under Uncertainty". In: *Robotics and Autonomous Systems* 133 (2020), p. 103629.

[14] Daniel Tihanyi, Yimeng Lu, Orcun Karaca, and Maryam Kamgarpour. "Multi-Robot Task Allocation for Safe Planning under Dynamic Uncertainties". In: *arXiv preprint arXiv:2103.01840* (2021).

[15] Frits de Nijs, Matthijs Spaan, and Mathijs de Weerdt. "Preallocation and Planning Under Stochastic Resource Constraints". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[16] Kun Deng, Yushan Chen, and Calin Belta. "An Approximate Dynamic Programming Approach to Multiagent Persistent Monitoring in Stochastic Environments with Temporal Logic Constraints". In: *IEEE Transactions on Automatic Control* 62.9 (2017), pp. 4549–4563.

[17] Craig Boutilier. "Planning, Learning and Coordination in Multiagent Decision Processes". In: *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*. 1996, pp. 195–210.

[18] Ariel Felner, Roni Stern, Solomon Eyal Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan Sturtevant, Glenn Wagner, and Pavel Surynek. "Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges". In: *Proceedings of the Tenth Annual Symposium on Combinatorial Search (SoCS)*. 2017, pp. 29–37.

[19] Masoumeh Mansouri, Bruno Lacerda, Nick Hawes, and Federico Pecora. "Multi-Robot Planning Under Uncertain Travel Times and Safety Constraints". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2019, pp. 478–484.

[20] Carlos Azevedo, Bruno Lacerda, Nick Hawes, and Pedro Lima. "Long-Run Multi-Robot Planning under Uncertain Action Durations for Persistent Tasks". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 4323–4328.

[21] Shiqi Zhang, Yuqian Jiang, Guni Sharon, and Peter Stone. "Multirobot Symbolic Planning Under Temporal Uncertainty". In: *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems. 2017, pp. 501–510.

[22] João Vicente Messias, Matthijs Spaan, and Pedro Lima. "GSMDPs for Multi-Robot Sequential Decision-Making". In: *Proceedings of the 27th AAAI Conference on Artificial Intelligence*. 2013, pp. 1408–1414.

[23] Charlie Street, Bruno Lacerda, Manuel Mühlig, and Nick Hawes. "Multi-Robot Planning Under Uncertainty with Congestion-Aware Models". In: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2020.

[24] Charlie Street, Sebastian Pütz, Manuel Mühlig, Nick Hawes, and Bruno Lacerda. "Congestion-Aware Policy Synthesis for Multirobot Systems". In: *IEEE Transactions on Robotics* (2021).

[25] Charlie Street, Bruno Lacerda, Michal Staniaszek, Manuel Mühlig, and Nick Hawes. "Context-Aware Modelling for Multi-Robot Systems Under Uncertainty". In: *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2022.

[26] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. "Model-Checking Continuous-Time Markov Chains". In: *ACM Transactions on Computational Logic (TOCL)* 1.1 (2000), pp. 162–170.

[27] Charlie Street, Bruno Lacerda, Manuel Mühlig, and Nick Hawes. "Right Place, Right Time: Proactive Multi-Robot Task Allocation Under Spatiotemporal Uncertainty". In: *Under Review*. 2022.

[28] Sertac Karaman and Emilio Frazzoli. "Sampling-Based Algorithms for Optimal Motion Planning". In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894.

[29] Mike Phillips and Maxim Likhachev. "SIPP: Safe Interval Path Planning for Dynamic Environments". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 5628–5635.

[30] Marlin Strub and Jonathan Gammell. "Adaptively Informed Trees (AIT*) and Effort Informed Trees (EIT*): Asymmetric Bidirectional Sampling-Based Path Planning". In: *The International Journal of Robotics Research (IJRR)* 41.4 (2022), pp. 390–417.

[31] Wil Thomason, Marlin Strub, and Jonathan Gammell. "Task and Motion Informed Trees (TMIT*): Almost-Surely Asymptotically Optimal Integrated Task and Motion Planning". In: *IEEE Robotics and Automation Letters (RA-L)* 7.4 (2022), pp. 11370–11377.

[32] Zuoyue Li, Jan Dirk Wegner, and Aurélien Lucchi. "Topological Map Extraction from Overhead Images". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1715–1724.

[33] Jesús Hernández, Jesús Savage, Marco Negrete, Luis Contreras, Carlos Sarmiento, Oscar Fuentes, and Hiroyuki Okada. "Sparse-Map: Automatic Topological Map Creation via Unsupervised Learning Techniques". In: *Advanced Robotics* (2022), pp. 1–11.

[34] Bruno Lacerda, Fatma Faruq, David Parker, and Nick Hawes. "Probabilistic Planning with Formal Performance Guarantees for Mobile Service Robots". In: *The International Journal of Robotics Research* 38.9 (2019), pp. 1098–1123.

[35] Matthew Budd, Bruno Lacerda, Paul Duckworth, Andrew West, Barry Lennox, and Nick Hawes. "Markov Decision Processes with Unknown State Feature Values for Safe Exploration using Gaussian Processes". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 7344–7350.

[36] M Wells, Zachary Kingston, Morteza Lahijanian, Lydia E Kavraki, and Moshe Y Vardi. "Finite-Horizon Synthesis for Probabilistic Manipulation Domains". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 6336–6342.

[37] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Morgan & Claypool Publishers, 2012.

[38] Dimitri P Bertsekas and John N Tsitsiklis. "An Analysis of Stochastic Shortest Path Problems". In: *Mathematics of Operations Research* 16.3 (1991), pp. 580–595.

[39] Sergio Gómez, Alexandre Arenas, Javier Borge-Holthoefer, Sandro Meloni, and Yamir Moreno. "Discrete-Time Markov Chain Approach to Contact-Based Disease Spreading in Complex Networks". In: *Europhysics Letters (EPL)* 89.3 (2010), p. 38009.

[40] Marta Kwiatkowska, Gethin Norman, and David Parker. "Stochastic Model Checking". In: *International School on Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM)*. Springer, 2007, pp. 220–270.

[41] Rachid El Abdouni Khayari, Ramin Sadre, and Boudewijn R Haverkort. "Fitting World-Wide Web Request Traces with the EM-Algorithm". In: *Performance Evaluation* 52.2-3 (2003), pp. 175–191.

[42] Tianhui Meng, Xiaofan Li, Sha Zhang, and Yubin Zhao. "A Hybrid Secure Scheme for Wireless Sensor Networks Against Timing Attacks using Continuous-Time Markov Chain and Queueing Model". In: *Sensors* 16.10 (2016), p. 1606.

[43] Ben Othman Jalel, Vèque Véronique, et al. "Continuous Time Markov Chain Traffic Model for Urban Environments". In: *Proceedings of the GLOBECOM IEEE Global Communications Conference*. IEEE. 2020, pp. 1–6.

[44] David F Anderson and Thomas G Kurtz. "Continuous Time Markov Chain Models for Chemical Reaction Networks". In: *Design and Analysis of Biomolecular Circuits*. Springer, 2011, pp. 3–42.

[45] MJ Faddy and SI0966 McClean. "Analysing Data on Lengths of Stay of Hospital Patients using Phase-Type Distributions". In: *Applied Stochastic Models in Business and Industry* 15.4 (1999), pp. 311–317.

[46] Reza Pulungan and Holger Hermanns. "Transient Analysis of CTMCs: Uniformization or Matrix Exponential?" In: *IAENG International Journal of Computer Science* 45.2 (2018), pp. 267–274.

[47] Bennett L Fox and Peter W Glynn. "Computing Poisson Probabilities". In: *Communications of the ACM* 31.4 (1988), pp. 440–445.

[48] M. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of Probabilistic Real-time Systems". In: *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV)*. Springer, 2011, pp. 585–591.

[49] Peter Buchholz, Jan Kriege, and Iryna Felko. *Input Modeling with Phase-Type Distributions and Markov Models: Theory and Applications*. Springer, 2014.

[50] Colm Art O'Cinneide. "Phase-Type Distributions: Open Problems and a Few Properties". In: *Stochastic Models* 15.4 (1999), pp. 731–757.

[51] Axel Thummler, Peter Buchholz, and Miklos Telek. "A Novel Approach for Phase-Type Fitting with the EM Algorithm". In: *IEEE Transactions on Dependable and Secure Computing* 3.3 (2006), pp. 245–258.

[52] Håkan LS Younes and Reid G Simmons. "Solving Generalized Semi-Markov Decision Processes using Continuous Phase-Type Distributions". In: *Proceedings of the 19th AAAI Conference on Artificial Intelligence*. 2004, pp. 742–747.

[53] George Casella and Roger L Berger. *Statistical Inference*. Cengage Learning, 1990.

[54] Philipp Reinecke, Tilman Krauß, and Katinka Wolter. "Cluster-Based Fitting of Phase-Type Distributions to Empirical Data". In: *Computers & Mathematics with Applications* 64.12 (2012), pp. 3840–3851.

[55] Colin Paterson and Radu Calinescu. "Accurate Analysis of Quality Properties of Software with Observation-Based Markov Chain Refinement". In: *Proceedings of the IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2017, pp. 121–130.

[56] Christian Eisentraut, Holger Hermanns, and Lijun Zhang. "On Probabilistic Automata in Continuous Time". In: *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science*. IEEE, 2010, pp. 342–351.

[57] Hassan Hatefi and Holger Hermanns. "Model Checking Algorithms for Markov Automata". In: *Electronic Communications of the EASST* 53 (2012).

[58] Dennis Guck, Hassan Hatefi, H Hermanns, Joost P Katoen, and Mark Timmer. "Analysis of Timed and Long-Run Objectives for Markov Automata". In: *Logical Methods in Computer Science* 10.3 (2014).

[59] Dennis Guck, Mark Timmer, Hassan Hatefi, Enno Ruijters, and Mariëlle Stoelinga. "Modelling and Analysis of Markov Reward Automata". In: *Proceedings of the International Symposium on Automated Technology for Verification and Analysis*. Springer. 2014, pp. 168–184.

[60] Tim Quatmann, Sebastian Junges, and Joost-Pieter Katoen. "Markov Automata with Multiple Objectives". In: *Proceedings of the International Conference on Computer Aided Verification*. Springer. 2017, pp. 140–159.

[61] Yuan H Wang. "On the Number of Successes in Independent Trials". In: *Statistica Sinica* (1993), pp. 295–312.

[62] Yili Hong. "On Computing the Distribution Function for the Poisson Binomial Distribution". In: *Computational Statistics & Data Analysis* 59 (2013), pp. 41–51.

[63] A Yu Volkova. "A Refinement of the Central Limit Theorem for Sums of Independent Random Indicators". In: *Theory of Probability & Its Applications* 40.4 (1996), pp. 791–794.

[64] R. Barlow and Klaus Heidtmann. "Computing k-out-of-n System Reliability". In: *IEEE Transactions on Reliability* (1984), pp. 322–323.

[65] William Biscarri, Sihai Dave Zhao, and Robert J Brunner. "A Simple and Fast Method for Computing the Poisson Binomial Distribution Function". In: *Computational Statistics & Data Analysis* 122 (2018), pp. 92–100.

[66] Jin Soo Park, Brian Tsang, Harel Yedidsion, Garrett Warnell, Daehyun Kyoung, Peter Stone, and AI Sony. "Learning to Improve Multi-Robot Hallway Navigation". In: *Proceedings of the Conference on Robot Learning*. 2020.

[67] David Silver. "Cooperative Pathfinding". In: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. AAAI Press, 2005, pp. 117–122.

[68] Glenn Wagner and Howie Choset. "Subdimensional Expansion for Multirobot Path Planning". In: *Artificial Intelligence* 219 (2015), pp. 1–24.

[69] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. "The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding". In: *Artificial intelligence* 195 (2013), pp. 470–495.

[70] Thayne T Walker, Nathan R Sturtevant, and Ariel Felner. "Extended Increasing Cost Tree Search for Non-Unit Cost Domains." In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*. 2018, pp. 534–540.

[71] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. "Conflict-Based Search for Optimal Multi-Agent Pathfinding". In: *Artificial Intelligence* 219 (2015), pp. 40–66.

[72] Eli Boyarski, Ariel Felner, Daniel Harabor, Peter J Stuckey, Liron Cohen, Jiaoyang Li, and Sven Koenig. "Iterative-Deepening Conflict-Based Search". In: *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence (IJCAI)*. 2021, pp. 4084–4090.

[73] Jiaoyang Li, Daniel Harabor, Peter J Stuckey, Ariel Felner, Hang Ma, and Sven Koenig. "Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search". In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. Vol. 29. 2019, pp. 279–283.

[74] Glenn Wagner and Howie Choset. "Path Planning for Multiple Agents Under Uncertainty". In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS)*. 2017, pp. 577–585.

[75] Rongye Shi, Peter Steenkiste, and Manuela M Veloso. "SC-M*: A Multi-Agent Path Planning Algorithm with Soft-Collision Constraint on Allocation of Common Resources". In: *Applied Sciences* 9.19 (2019).

[76] Tomer Shahar, Shashank Shekhar, Dor Atzmon, Abdallah Saffidine, Brendan Juba, and Roni Stern. "Safe Multi-Agent Pathfinding with Time Uncertainty". In: *Journal of Artificial Intelligence Research* 70 (2021), pp. 923–954.

[77] Anton Andreychuk, Konstantin Yakovlev, Dor Atzmon, and Roni Stern. "Multi-Agent Pathfinding with Continuous Time". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*. 2019, pp. 39–45.

[78] Anton Andreychuk, Konstantin Yakovlev, Eli Boyarski, and Roni Stern. "Improving Continuous-time Conflict Based Search". In: *Proceedings of the International Symposium on Combinatorial Search (SoCS)*. Vol. 12. 1. 2021, pp. 145–146.

[79] Anton Andreychuk. "Multi-Agent Path Finding with Kinematic Constraints via Conflict Based Search". In: *Proceedings of the Russian Conference on Artificial Intelligence*. Springer. 2020, pp. 29–45.

[80] Jie Wang, Tao Wu, and Meiping Shi. "Persistent Execution of Continuous Time Multi-Robot Pathfinding". In: *Proceedings of the China Automation Congress (CAC)*. IEEE. 2021, pp. 6964–6969.

[81] Pavel Surynek. "Multi-Agent Path Finding with Continuous Time and Geometric Agents Viewed through Satisfiability Modulo Theories (SMT)". In: *Proceedings of the Twelfth Annual Symposium on Combinatorial Search (SoCS)*. 2019, pp. 200–201.

[82] Oriana Peltzer, Kyle Brown, Mac Schwager, Mykel J Kochenderfer, and Martin Sehr. "STT-CBS: A Conflict-Based Search Algorithm for Multi-Agent Path Finding with Stochastic Travel Times". In: *arXiv preprint arXiv:2004.08025* (2020).

[83] Ofra Amir, Guni Sharon, and Roni Stern. "Multi-Agent Pathfinding as a Combinatorial Auction". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[84] Anna Gautier, Alex Stephens, Bruno Lacerda, Nick Hawes, and Michael Woolridge. "Negotiated Path Planning for Non-Cooperative Multi-Robot Systems". In: *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2022.

[85] Richard Bellman. "Dynamic Programming". In: *Science* 153.3731 (1966), pp. 34–37.

[86] Ronald A Howard. *Dynamic Programming and Markov Processes*. John Wiley, 1960.

[87] Blai Bonet and Hector Geffner. "Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming". In: *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS)*. 2003, pp. 12–21.

[88] H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. "Bounded Real-Time Dynamic Programming: RTDP with Monotone Upper Bounds and Performance Guarantees". In: *Proceedings of the 22nd International Conference on Machine learning (ICML)*. 2005, pp. 569–576.

[89] Sung Wook Yoon, Alan Fern, and Robert Givan. "FF-Replan: A Baseline for Probabilistic Planning." In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 2007, pp. 352–359.

[90] Levente Kocsis and Csaba Szepesvári. "Bandit Based Monte-Carlo Planning". In: *Proceedings of the European Conference on Machine Learning*. Springer. 2006, pp. 282–293.

[91] Paul Duckworth, Bruno Lacerda, and Nick Hawes. "Time-Bounded Mission Planning in Time-Varying Domains with Semi-MDPs and Gaussian Processes". In: *Proceedings of the 4th Annual Conference on Robot Learning (CoRL)*. 2020.

[92] Min Chen, Emilio Frazzoli, David Hsu, and Wee Sun Lee. "POMDP-Lite for Robust Robot Planning Under Uncertainty". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 5427–5433.

[93] Graeme Best, Oliver M Cliff, Timothy Patten, Ramgopal R Mettu, and Robert Fitch. "Dec-MCTS: Decentralized Planning for Multi-Robot Active Perception". In: *The International Journal of Robotics Research* 38.2-3 (2019), pp. 316–337.

[94] Andrey Kolobov, Daniel S Weld, et al. "LRTDP Versus UCT for Online Probabilistic Planning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2012.

[95] Bruno Lacerda, David Parker, and Nick Hawes. "Multi-Objective Policy Generation for Mobile Robots under Probabilistic Time-Bounded Guarantees". In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS)*. 2017.

[96] Joris Scharpff, Diederik Roijers, Frans Oliehoek, Matthijs Spaan, and Mathijs de Weerdt. "Solving Transition-Independent Multi-Agent MDPs with Sparse Interactions". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[97] Frits De Nijs, Erwin Walraven, Mathijs De Weerdt, and Matthijs Spaan. "Constrained Multiagent Markov Decision Processes: A Taxonomy of Problems and Algorithms". In: *Journal of Artificial Intelligence Research* 70 (2021), pp. 955–1001.

[98] Fatma Faruq, David Parker, Bruno Lacerda, and Nick Hawes. "Simultaneous Task Allocation and Planning Under Uncertainty". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 3559–3564.

[99] Michael L Littman. "Markov Games as a Framework for Multi-Agent Reinforcement Learning". In: *Machine Learning Proceedings*. Elsevier, 1994, pp. 157–163.

[100] Michael L Littman. "Friend-or-Foe Q-learning in General-Sum Games". In: *Proceedings of the Eighteenth International Conference on Machine Learning*. 2001, pp. 322–328.

[101] Matthijs T.J. Spaan and Francesco S. Melo. "Interaction-Driven Markov Games for Decentralized Multiagent Planning Under Uncertainty". In: *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[102] Aleksander Czechowski and Frans A. Oliehoek. "Decentralized MCTS via Learned Teammate Models". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI)*. 2021, pp. 81–88.

[103] Daniel Claes, Philipp Robbel, Frans Oliehoek, Karl Tuyls, Daniel Hennes, and Wiebe Van der Hoek. "Effective Approximations for Multi-Robot Coordination in Spatially Distributed Tasks". In: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems. 2015, pp. 881–890.

[104] Daniel Claes, Frans Oliehoek, Hendrik Baier, and Karl Tuyls. "Decentralised Online Planning for Multi-Robot Warehouse Commissioning". In: *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems. 2017, pp. 492–500.

[105] Andrea Bajcsy, Sylvia L Herbert, David Fridovich-Keil, Jaime F Fisac, Sampada Deglurkar, Anca D Dragan, and Claire J Tomlin. "A Scalable Framework for Real-Time Multi-Robot, Multi-Human Collision Avoidance". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 936–943.

[106] Maren Bennewitz, Wolfram Burgard, and Sebastian Thrun. "Optimizing Schedules for Prioritized Path Planning of Multi-Robot Systems". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2001, pp. 271–276.

[107] Jur P Van Den Berg and Mark H Overmars. "Prioritized Motion Planning for Multiple Robots". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2005, pp. 430–435.

[108] Wenying Wu, Subhrajit Bhattacharya, and Amanda Prorok. "Multi-Robot Path Deconfliction through Prioritization by Path Prospects". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 9809–9815.

[109] Maren Bennewitz, Wolfram Burgard, and Sebastian Thrun. "Finding and Optimizing Solvable Priority Schemes for Decoupled Path Planning Techniques for Teams of Mobile Robots". In: *Robotics and Autonomous Systems* 41.2-3 (2002), pp. 89–99.

[110] Wenying Wu, Subhrajit Bhattacharya, and Amanda Prorok. "Multi-Robot Path Deconfliction through Prioritization by Path Prospects". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 9809–9815.

[111] Francisco S Melo and Manuela Veloso. "Decentralized MDPs with Sparse Interactions". In: *Artificial Intelligence* 175.11 (2011), pp. 1757–1789.

[112] Francisco S Melo and Manuela Veloso. "Heuristic Planning for Decentralized MDPs with Sparse Interactions". In: *Distributed Autonomous Robotic Systems*. Springer, 2013, pp. 329–343.

[113] Bin Yang, Chenjuan Guo, Christian S Jensen, Manohar Kaul, and Shuo Shang. "Stochastic Skyline Route Planning Under Time-Varying Uncertainty". In: *Proceedings of the 30th IEEE International Conference on Data Engineering*. IEEE. 2014, pp. 136–147.

[114] Filip Surma, Tomasz Piotr Kucner, and Masoumeh Mansouri. "Multiple Robots Avoid Humans To Get the Jobs Done: An Approach to Human-aware Task Allocation". In: *Proceedings of the European Conference on Mobile Robots (ECMR)*. IEEE. 2021, pp. 1–6.

[115] Mathijs M de Weerdt, Sebastian Stein, Enrico H Gerding, Valentin Robu, and Nicholas R Jennings. "Intention Aware Routing of Electric Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* 17.5 (2015), pp. 1472–1482.

[116] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* 112.1-2 (1999), pp. 181–211.

[117] Xianping Guo and Onésimo Hernández-Lerma. *Continuous-Time Markov Decision Processes: Theory and Applications*. Springer-Verlag Berlin Heidelberg, 2009.

[118] AI Medina Ayala, Sean B Andersson, and Calin Belta. "Probabilistic Control from Time-Bounded Temporal Logic Specifications in Dynamic Environments". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012, pp. 4705–4710.

[119] Zhengyu Yin and Milind Tambe. "Continuous Time Planning for Multiagent Teams with Temporal Constraints". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[120] Prasadarao Kakumanu. "Relation Between Continuous and Discrete Time Markovian Decision Problems". In: *Naval Research Logistics Quarterly* 24.3 (1977), pp. 431–439.

[121] Yuliya Butkova, Ralf Wimmer, and Holger Hermanns. "Long-Run Rewards for Markov Automata". In: *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer. 2017, pp. 188–203.

[122] Gianfranco Balbo. "Introduction to Generalized Stochastic Petri Nets". In: *Proceedings of the International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer. 2007, pp. 83–131.

[123] Hugo Costelha and Pedro Lima. "Robot Task Plan Representation by Petri Nets; Modelling, Identification, Analysis and Execution". In: *Autonomous Robots* 33.4 (2012), pp. 337–360.

[124] Christian Eisentraut, Holger Hermanns, Joost-Pieter Katoen, and Lijun Zhang. "A Semantics for Every GSPN". In: *Proceedings of the 34th International Conference on Applications and Theory of Petri Nets and Concurrency (Petri Nets)*. Springer. 2013.

[125] Grigory Neustroev, Mathijs de Weerdt, and Remco Verzijlbergh. "Discovery of Optimal Solution Horizons in Non-Stationary Markov Decision Processes with Unbounded Rewards". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 29. 2019, pp. 292–300.

[126] Justin A Boyan and Michael L Littman. "Exact Solutions to Time-Dependent MDPs". In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. MIT Press, 2000, pp. 1026–1032.

[127] Emmanuel Rachelson, Patrick Fabiani, and Frédérick Garcia. "TiMDPpoly: An Improved Method for Solving Time-Dependent MDPs". In: *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2009, pp. 796–799.

[128] Lantao Liu and Gaurav S Sukhatme. "A Solution to Time-Varying Markov Decision Processes". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1631–1638.

[129] Junhong Xu, Kai Yin, and Lantao Liu. "Reachable Space Characterization of Markov Decision Processes with Time Variability". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2019.

[130] Abdullah Al Redwan Newaz, Swarat Chaudhuri, and Lydia E Kavraki. "Monte-Carlo Policy Synthesis in POMDPs with Quantitative and Qualitative Objectives". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2019.

[131] Jesus Capitan, Matthijs TJ Spaan, Luis Merino, and Anibal Ollero. "Decentralized Multi-Robot Cooperation with Auctioned POMDPs". In: *The International Journal of Robotics Research* 32.6 (2013), pp. 650–671.

[132] Guy Shani, Joelle Pineau, and Robert Kaplow. "A Survey of Point-Based POMDP Solvers". In: *Autonomous Agents and Multi-Agent Systems* 27.1 (2013), pp. 1–51.

[133] David Silver and Joel Veness. "Monte-Carlo Planning in Large POMDPs". In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. Vol. 23. 2010.

[134] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. "The Complexity of Decentralized Control of Markov Decision Processes". In: *Mathematics of Operations Research* 27.4 (2002), pp. 819–840.

[135] Christopher Amato, George Konidaris, Gabriel Cruz, Christopher A Maynor, Jonathan P How, and Leslie P Kaelbling. "Planning for Decentralized Control of Multiple Robots Under Uncertainty". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 1241–1248.

[136] Shayegan Omidshafiei, Ali-Akbar Agha-Mohammadi, Christopher Amato, Shih-Yuan Liu, Jonathan P How, and John Vian. "Decentralized Control of Multi-Robot Partially Observable Markov Decision Processes using Belief Space Macro-Actions". In: *The International Journal of Robotics Research* 36.2 (2017), pp. 231–258.

[137] Bruno Floriano, Geovany A Borges, and Henrique Ferreira. "Planning for Decentralized Formation Flight of UAV Fleets in Uncertain Environments with Dec-POMDP". In: *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2019, pp. 563–568.

[138] Frans Oliehoek, Stefan Witwicki, and Leslie Kaelbling. "Influence-Based Abstraction for Multiagent Systems". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 26. 1. 2012, pp. 1422–1428.

[139] Frans Oliehoek, Stefan Witwicki, and Leslie Kaelbling. "A Sufficient Statistic for Influence in Structured Multiagent Environments". In: *Journal of Artificial Intelligence Research* 70 (2021), pp. 789–870.

[140] Christopher Amato, George Konidaris, Leslie P Kaelbling, and Jonathan P How. "Modeling and Planning with Macro-Actions in Decentralized POMDPs". In: *Journal of Artificial Intelligence Research* 64 (2019), pp. 817–859.

[141] Christopher Amato, George D Konidaris, and Leslie P Kaelbling. "Planning with Macro-Actions in Decentralized POMDPs". In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems. 2014, pp. 1273–1280.

[142] Christopher Amato, George Konidaris, Ariel Anders, Gabriel Cruz, Jonathan P How, and Leslie P Kaelbling. "Policy Search for Multi-Robot Coordination Under Uncertainty". In: *The International Journal of Robotics Research* 35.14 (2016), pp. 1760–1778.

[143] Christopher Amato. "Decision-Making Under Uncertainty in Multi-Agent and Multi-Robot Systems: Planning and Learning". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2018, pp. 5662–5666.

[144] Erwin Walraven and Matthijs TJ Spaan. "Column Generation Algorithms for Constrained POMDPs". In: *Journal of Artificial Intelligence Research* 62 (2018), pp. 489–533.

[145] Frits De Nijs, Matthijs Spaan, and Mathijs de Weerdt. "Best-response planning of thermostatically controlled loads under power constraints". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2015.

[146] Frits de Nijs and Peter J Stuckey. "Risk-Aware Conditional Replanning for Globally Constrained Multi-Agent Sequential Decision Making". In: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2020, pp. 303–311.

[147] Frits De Nijs, Erwin Walraven, Mathijs de Weerdt, and Matthijs Spaan. "Bounding the Probability of Resource Constraint Violations in Multi-Agent MDPs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[148] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An Introduction*. MIT press, 2018.

[149] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. "MOReL: Model-Based Offline Reinforcement Learning". In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. 2020, pp. 21810–21823.

[150] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "Deep Reinforcement Learning: A Brief Survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[151] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3389–3396.

[152] Siddhant Gangapurwala, Mathieu Geisert, Romeo Orsolino, Maurice Fallon, and Ioannis Havoutis. "RLOC: Terrain-Aware Legged Locomotion Using Reinforcement Learning and Optimal Control". In: *IEEE Transactions on Robotics* (2022).

[153] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning". In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2018, pp. 4295–4304.

[154] Minghuan Liu, Menghui Zhu, and Weinan Zhang. "Goal-Conditioned Reinforcement Learning: Problems and Solutions". In: 2022.

[155] Nathan Koenig and Andrew Howard. "Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sendai, Japan, 2004, pp. 2149–2154.

[156] Olivier Michel. "Cyberbotics Ltd. Webots™: Professional Mobile Robot Simulation". In: *International Journal of Advanced Robotic Systems* 1.1 (2004), p. 5.

[157] E. Rohmer, S. P. N. Singh, and M. Freese. "CoppeliaSim (formerly V-REP): A Versatile and Scalable Robot Simulation Framework". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013.

[158] Jemin Hwangbo, Joonho Lee, and Marco Hutter. "Per-Contact Iteration Method for Solving Contact Dynamics". In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 895–902.

[159] Jens Happe and Jean Berger. "CoUAV: A Multi-UAV Cooperative Search Path Planning Simulation Environment". In: *Proceedings of the Summer Computer Simulation Conference (SCSC)*. 2010, pp. 86–93.

[160] Tim Bakker, Garrett L Ward, Siva T Patibandla, and Robert H Klenke. "RAMS: A Fast, Low-Fidelity, Multiple Agent Discrete-Event Simulator". In: *Proceedings of the Summer Computer Simulation Conference (SCSC)*. 2013, pp. 1–10.

[161] Maria Hybinette, Eileen Kraemer, Yin Xiong, Glenn Matthews, and Jaim Ahmed. "SASSY: A Design for a Scalable Agent-Based Simulation System using a Distributed Discrete Event Infrastructure". In: *Proceedings of the Winter Simulation Conference*. IEEE. 2006, pp. 926–933.

[162] Fei Gao and Missy Cummings. "Using Discrete Event Simulation to Model Multi-Robot Multi-Operator Teamwork". In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 56. 1. SAGE Publications Sage CA: Los Angeles, CA. 2012, pp. 2093–2097.

[163] Bruno Damas and Pedro Lima. "Stochastic Discrete Event Model of a Multi-Robot Team Playing an Adversarial Game". In: *Proceedings of the IFAC/EURON Symposium on Intelligent Autonomous Vehicles*. Vol. 37. 8. Elsevier, 2004, pp. 974–979.

[164] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[165] Rajeev Alur, Thomas A Henzinger, and Orna Kupferman. "Alternating-Time Temporal Logic". In: *Journal of the ACM (JACM)* 49.5 (2002), pp. 672–713.

[166] Taolue Chen and Jian Lu. "Probabilistic Alternating-Time Temporal Logic and Model Checking Algorithm". In: *Proceedings of the Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. Vol. 2. IEEE. 2007, pp. 35–39.

[167] Natasha Alechina, Brian Logan, Nguyen Hoang Nga, and Abdur Rakib. "Resource-Bounded Alternating-Time Temporal Logic". In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2010, pp. 481–488.

[168] Hoang Nga Nguyen and Abdur Rakib. "A Probabilistic Logic for Resource-Bounded Multi-Agent Systems". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*. 2019, pp. 521–527.

[169] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. "Bounded Model Checking". In: *Advances in Computers* 58 (2003).

[170] Wojciech Penczek and Alessio Lomuscio. "Verifying Epistemic Properties of Multi-Agent Systems via Bounded Model Checking". In: *Fundamenta Informaticae* 55.2 (2003), pp. 167–185.

[171] Mika Cohen, Mads Dam, Alessio Lomuscio, and Francesco Russo. "Abstraction in Model Checking Multi-Agent Systems". In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent System (AAMAS)*. 2009, pp. 945–952.

[172] Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger. "Automatic Verification of Multi-Agent Systems in Parameterised Grid-Environments". In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2016, pp. 1190–1199.

[173] Panagiotis Kouvaros and Alessio Lomuscio. "Automatic Verification of Parameterised Interleaved Multi-Agent Systems". In: *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 861–868.

[174] Panagiotis Kouvaros and Alessio Lomuscio. "Verifying Fault-tolerance in Parameterised Multi-Agent Systems". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2017, pp. 288–294.

[175] Axel Legay, Benoît Delahaye, and Saddek Bensalem. "Statistical Model Checking: An Overview". In: *Proceedings of the International Conference on Runtime Verification*. 2010, pp. 122–135.

[176] Benjamin Herd, Simon Miles, Peter McBurney, and Michael Luck. "Quantitative Analysis of Multiagent Systems through Statistical Model Checking". In: *Proceedings of the International Workshop on Engineering Multi-Agent Systems*. Springer. 2015, pp. 109–130.

[177] Yuliya Butkova, Arnd Hartmanns, and Holger Hermanns. "A Modest Approach to Markov Automata". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 31.3 (2021), pp. 1–34.

[178] Alessio Lomuscio and Franco Raimondi. "MCMAS: A Model Checker for Multi-Agent Systems". In: *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2006, pp. 450–454.

[179] Michael Wooldridge, Michael Fisher, Marc-Philippe Huget, and Simon Parsons. "Model Checking Multi-Agent Systems with MABLE". In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2002, pp. 952–959.

[180] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. "PRISM-Games 3.0: Stochastic Game Verification with Concurrency, Equilibria and Time". In: *Proceedings of the 32nd International Conference on Computer Aided Verification (CAV)*. Springer, 2020, pp. 475–487.

[181] Matt Webster, David Western, Dejanira Araiza-Illan, Clare Dixon, Kerstin Eder, Michael Fisher, and Anthony G Pipe. "A Corroborative Approach to Verification and Validation of Human–Robot Teams". In: *The International Journal of Robotics Research* 39.1 (2020), pp. 73–99.

[182] Georg Friedrich Schuppe and Jana Tumova. "Decentralized Multi-Agent Strategy Synthesis under LTLf Specifications via Exchange of Least-Limiting Advisers". In: *Proceedings of the International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE. 2021, pp. 119–127.

[183] Alexandros Nikou, Dimitris Boskos, Jana Tumova, and Dimos V Dimarogonas. "On the Timed Temporal Logic Planning of Coupled Multi-Agent Systems". In: *Automatica* 97 (2018), pp. 339–345.

[184] Tichakorn Wongpiromsarn, Alphan Ulusoy, Calin Belta, Emilio Frazzoli, and Daniela Rus. "Incremental Synthesis of Control Policies for Heterogeneous Multi-Agent Systems with Linear Temporal Logic Specifications". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 5011–5018.

[185] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. "Improving Multi-Robot Behavior using Learning-Based Receding Horizon Task Allocation". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2018.

[186] David Gundana and Hadas Kress-Gazit. "Event-Based Signal Temporal Logic Synthesis for Single and Multi-Robot Tasks". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3687–3694.

[187] Mohit Srinivasan, Samuel Coogan, and Magnus Egerstedt. "Control of Multi-Agent Systems with Finite Time Control Barrier Certificates and Temporal Logic". In: *Proceedings of the IEEE Conference on Decision and Control (CDC)*. IEEE. 2018, pp. 1991–1996.

[188] Sertac Karaman and Emilio Frazzoli. "Vehicle Routing Problem with Metric Temporal Logic Specifications". In: *Proceedings of the 47th IEEE Conference on Decision and Control (CDC)*. IEEE. 2008, pp. 3953–3958.

[189] Zhiyu Liu, Jin Dai, Bo Wu, and Hai Lin. "Communication-Aware Motion Planning for Multi-Agent Systems from Signal Temporal Logic Specifications". In: *Proceedings of the American Control Conference (ACC)*. IEEE. 2017, pp. 2516–2521.

[190] Yiannis Kantaros and Michael M Zavlanos. "Sampling-Based Optimal Control Synthesis for Multirobot Systems Under Global Temporal Tasks". In: *IEEE Transactions on Automatic Control* 64.5 (2018), pp. 1916–1931.

[191] Yiannis Kantaros and Michael M Zavlanos. "Temporal Logic Optimal Control for Large-Scale Multi-Robot Systems: 10^{400} states and beyond". In: *Proceedings of the IEEE Conference on Decision and Control (CDC)*. IEEE. 2018, pp. 2519–2524.

[192] Fernando S Barbosa, Lars Lindemann, Dimos V Dimarogonas, and Jana Tumova. "Integrated Motion Planning and Control Under Metric Interval Temporal Logic Specifications". In: *Proceedings of the 18th European Control Conference (ECC)*. IEEE. 2019, pp. 2042–2049.

[193] Alex Rutherford, Paul Duckworth, Nick Hawes, and Bruno Lacerda. "Motion Planning in Uncertain Environments with Rapidly-Exploring Random Markov Decision Processes". In: *Proceedings of the European Conference on Mobile Robots (ECMR)*. IEEE, pp. 1–6.

[194] Brian P Gerkey and Maja J Matarić. "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems". In: *The International Journal of Robotics Research* 23.9 (2004), pp. 939–954.

[195] Harold W Kuhn. "The Hungarian Method for the Assignment Problem". In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 83–97.

[196] Smriti Chopra, Giuseppe Notarstefano, Matthew Rice, and Magnus Egerstedt. "A Distributed Version of the Hungarian Method for Multirobot Assignment". In: *IEEE Transactions on Robotics* 33.4 (2017), pp. 932–947.

[197] Brian P Gerkey and Maja J Mataric. "Sold!: Auction Methods for Multirobot Coordination". In: *IEEE Transactions on Robotics and Automation* 18.5 (2002), pp. 758–768.

[198] Michael M Zavlanos, Leonid Spesivtsev, and George J Pappas. "A Distributed Auction Algorithm for the Assignment Problem". In: *Proceedings of the 47th IEEE Conference on Decision and Control (CDC)*. IEEE. 2008, pp. 1212–1217.

[199] Liad Blumrosen and Noam Nisan. "Combinatorial Auctions". In: *Algorithmic Game Theory* 267 (2007), p. 300.

[200] Sven Koenig, Craig Tovey, Michail Lagoudakis, V Markakis, David Kempe, Pinar Keskinocak, Anton Kleywegt, Adam Meyerson, and Sonal Jain. "The Power of Sequential Single-Item Auctions for Agent Coordination". In: *Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence*. 2006, pp. 1625–1629.

[201] Adrian Schoenig and Maurice Pagnucco. "Evaluating Sequential Single-Item Auctions for Dynamic Task Allocation". In: *Proceedings of the Australasian Joint Conference on Artificial Intelligence*. Springer. 2010, pp. 506–515.

[202] Xiaoming Zheng, Sven Koenig, and Craig Tovey. "Improving Sequential Single-Item Auctions". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2006, pp. 2238–2244.

[203] Sven Koenig, Craig A Tovey, Xiaoming Zheng, and Ilgaz Sungur. "Sequential Bundle-Bid Single-Sale Auction Algorithms for Decentralized Control". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2007, pp. 1359–1365.

[204] Sven Koenig Xiaoming Zheng, Craig Tovey, Richard Borie, Philip Kilby, Vangelis Markakis, and Pinar Keskinocak. "Agent Coordination with Regret Clearing". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2008, p. 101.

[205] Ernesto Nunes and Maria Gini. "Multi-Robot Auctions for Allocation of Tasks with Temporal Constraints". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2015.

[206] Ernesto Nunes, Mitchell McIntire, and Maria Gini. "Decentralized Multi-Robot Allocation of Tasks with Temporal and Precedence Constraints". In: *Advanced Robotics* 31.22 (2017), pp. 1193–1207.

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

[208] Gautham P Das, Thomas M McGinnity, Sonya A Coleman, and Laxmidhar Behera. "A Distributed Task Allocation Algorithm for a Multi-Robot System in Healthcare Facilities". In: *Journal of Intelligent & Robotic Systems* 80.1 (2015), pp. 33–58.

[209] Amanda Whitbrook, Qinggang Meng, and Paul WH Chung. "A Novel Distributed Scheduling Algorithm for Time-Critical Multi-Agent Systems". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 6451–6458.

[210] Alessandro Farinelli, Luca Iocchi, and Daniele Nardi. "Distributed On-Line Dynamic Task Assignment for Multi-Robot Patrolling". In: *Autonomous Robots* 41.6 (2017), pp. 1321–1345.

[211] Maitreyi Nanjanath and Maria Gini. "Performance Evaluation of Repeated Auctions for Robust Task Execution". In: *Proceedings of the International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer. 2008, pp. 317–327.

[212] Zeynab Talebpour and Alcherio Martinoli. "Multi-Robot Coordination in Dynamic Environments Shared with Humans". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 4593–4600.

[213] Liu Lin and Zhiqiang Zheng. "Combinatorial Bids Based Multi-Robot Task Allocation Method". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE. 2005, pp. 1145–1150.

[214] Robert Zlot and Anthony Stentz. "Complex Task Allocation for Multiple Robots". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2005, pp. 1515–1522.

[215] Farouq Zitouni, Ramdane Maamri, and Saad Harous. "FA–QABC–MRTA: A Solution for Solving the Multi-Robot Task Allocation Problem". In: *Intelligent Service Robotics* 12.4 (2019), pp. 407–418.

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

[217] Drew Wicke, David Freelan, and Sean Luke. "Bounty Hunters and Multiagent Task Allocation". In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2015, pp. 387–394.

[218] Elango Murugappan, Nachiappan Subramanian, Shams Rahman, Mark Goh, and Hing Kai Chan. "Performance Analysis of Clustering Methods for Balanced Multi-Robot Task Allocations". In: *International Journal of Production Research* (2021), pp. 1–16.

[219] Xing Su, Minjie Zhang, and Quan Bai. "Coordination for Dynamic Weighted Task Allocation in Disaster Environments with Time, Space and Communication Constraints". In: *Journal of Parallel and Distributed Computing* 97 (2016), pp. 47–56.

[220] Nathan Lindsay, Russell K Buehling, and Liang Sun. "A Sequential Task Addition Distributed Assignment Algorithm for Multi-Robot Systems". In: *Journal of Intelligent & Robotic Systems* 102.2 (2021), pp. 1–12.

[221] Inmo Jang, Hyo-Sang Shin, and Antonios Tsourdos. "Anonymous Hedonic Game for Task Allocation in a Large-Scale Multiple Agent System". In: *IEEE Transactions on Robotics* 34.6 (2018), pp. 1534–1548.

[222] Haluk Bayram and H Işıl Bozma. "Coalition Formation Games for Dynamic Multirobot Tasks". In: *The International Journal of Robotics Research* 35.5 (2016), pp. 514–527.

[223] Chun Liu and Andreas Kroll. "Memetic Algorithms for Optimal Task Allocation in Multi-Robot Systems for Inspection Problems with Cooperative Tasks". In: *Soft Computing* 19.3 (2015), pp. 567–584.

[224] Martina Lippi and Alessandro Marino. "A Mixed-Integer Linear Programming Formulation for Human Multi-Robot Task Allocation". In: *Proceedings of the IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*. IEEE. 2021, pp. 1017–1023.

[225] Shushman Choudhury, Jayesh K Gupta, Mykel J Kochenderfer, Dorsa Sadigh, and Jeannette Bohg. "Dynamic Multi-Robot Task Allocation Under Uncertainty and Temporal Constraints". In: *Autonomous Robots* (2021), pp. 1–17.

[226] Amanda Prorok. "Redundant Robot Assignment on Graphs with Uncertain Edge Costs". In: *Distributed Autonomous Robotic Systems*. Springer, 2019, pp. 313–327.

[227] Matthew Malencia, Vijay Kumar, George Pappas, and Amanda Prorok. "Fair Robust Assignment Using Redundancy". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 4217–4224.

[228] Sameera S Ponda, Luke B Johnson, and Jonathan P How. "Distributed Chance-Constrained Task Allocation for Autonomous Multi-Agent Teams". In: *Proceedings of the American Control Conference (ACC)*. IEEE. 2012, pp. 4528–4533.

[229] Jean-François Cordeau and Gilbert Laporte. "The Dial-a-Ride Problem: Models and Algorithms". In: *Annals of Operations Research* 153.1 (2007), pp. 29–46.

[230] Marco Pavone, Nabhendra Bisnik, Emilio Frazzoli, and Volkan Isler. "A Stochastic and Dynamic Vehicle Routing Problem with Time Windows and Customer Impatience". In: *Mobile Networks and Applications* 14.3 (2009), pp. 350–364.

[231] Shaunak D Bopardikar, Stephen L Smith, and Francesco Bullo. "On Dynamic Vehicle Routing with Time Constraints". In: *IEEE Transactions on Robotics* 30.6 (2014), pp. 1524–1532.

[232] Hang Ma, Jiaoyang Li, T.K. Satish Kumar, and Sven Koenig. "Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks". In: *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 837–845.

[233] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph Durham, and Nora Ayanian. "Conflict-Based Search with Optimal Task Assignment". In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2018.

[234] Chongjie Zhang and Julie A Shah. "Co-Optimizating Multi-Agent Placement with Task Assignment and Scheduling". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2016, pp. 3308–3314.

[235] G Ayorkor Korsah, Balajee Kannan, Brett Browning, Anthony Stentz, and M Bernardine Dias. "xBots: An Approach to Generating and Executing Optimal Multi-Robot Plans with Cross-Schedule Dependencies". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012, pp. 115–122.

[236] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. "Simultaneous Task Allocation and Planning for Temporal Logic Goals in Heterogeneous Multi-Robot Systems". In: *The International Journal of Robotics Research* 37.7 (2018), pp. 818–838.

[237] Yuxiao Chen, Ugo Rosolia, and Aaron D Ames. "Decentralized Task and Path Planning for Multi-Robot Systems". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4337–4344.

[238] Fan Yang and Nilanjan Chakraborty. "Chance Constrained Simultaneous Path Planning and Task Assignment for Multiple Robots with Stochastic Path Costs". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 6661–6667.

[239] Richard Vaughan. "Massively Multi-Robot Simulation in Stage". In: *Swarm intelligence* 2.2-4 (2008), pp. 189–208.

[240] Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. "Feature Selection in Machine Learning: A New Perspective". In: *Neurocomputing* 300 (2018), pp. 70–79.

[241] Sung-Hyuk Cha. "Comprehensive Survey on Distance/Similarity Measures Between Probability Density Functions". In: *International Journal of Mathematical Models and Methods in Applied Sciences* 1.4 (2007).

[242] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V Goldman. "Transition-Independent Decentralized Markov Decision Processes". In: *Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2003, pp. 41–48.

[243] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. "Multi-Agent Reinforcement Learning: An Overview". In: *Innovations in Multi-Agent Systems and Applications-1* (2010), pp. 183–221.

[244] Georgios E Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J Pappas. "Temporal Logic Motion Planning for Dynamic Robots". In: *Automatica* 45.2 (2009), pp. 343–352.

[245] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. "Stochastic Dynamic Programming with Factored Representations". In: *Artificial Intelligence* 121.1-2 (2000), pp. 49–107.

[246] Adam Lipowski and Dorota Lipowska. "Roulette-Wheel Selection via Stochastic Acceptance". In: *Physica A: Statistical Mechanics and its Applications* 391.6 (2012), pp. 2193–2196.

[247] Sebastian Pütz, Jorge Santos Simón, and Joachim Hertzberg. "Move Base Flex: A Highly Flexible Navigation Framework for Mobile Robots". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 3416–3421.

[248] Frank J Massey Jr. "The Kolmogorov-Smirnov Test for Goodness of Fit". In: *Journal of the American Statistical Association* 46.253 (1951), pp. 68–78.

[249] Andrey Kolobov, Mausam, and Daniel S. Weld. "A Theory of Goal-Oriented MDPs with Dead Ends". In: *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI)*. 2012, pp. 438–447.

[250] Robert W Floyd. "Algorithm 97: Shortest Path". In: *Communications of the ACM* 5.6 (1962), p. 345.

[251] Peter Buchholz and Ingo Schulz. "Numerical Analysis of Continuous Time Markov Decision Processes over Finite Horizons". In: *Computers & Operations Research* 38.3 (2011), pp. 651–659.

[252] Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. "Continuous Upper Confidence Trees". In: *Proceedings of the International Conference on Learning and Intelligent Optimization*. Springer. 2011, pp. 433–445.

[253] Andrew W Moore and Christopher G Atkeson. "Prioritized Sweeping: Reinforcement Learning with Less Data and Less Time". In: *Machine learning* 13.1 (1993), pp. 103–130.

[254] Stefano Mariani, Giacomo Cabri, and Franco Zambonelli. "Coordination of Autonomous Vehicles: Taxonomy and survey". In: *ACM Computing Surveys (CSUR)* 54.1 (2021), pp. 1–33.

[255] Fei Xue, Hengliang Tang, Qinghua Su, and Tao Li. "Task Allocation of Intelligent Warehouse Picking System based on Multi-Robot Coalition". In: *KSII Transactions on Internet and Information Systems (TIIS)* 13.7 (2019), pp. 3566–3582.

[256] Helen Harman and Elizabeth I Sklar. "Multi-Agent Task Allocation for Fruit Picker Team Formation". In: *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2022, pp. 1618–1620.

[257] Odkhishig Ganbold, Kaustav Kundu, Haobin Li, and Wei Zhang. "A Simulation-Based Optimization Method for Warehouse Worker Assignment". In: *Algorithms* 13.12 (2020), p. 326.

[258] Andrey Glaschenko, Anton Ivaschenko, George Rzevski, and Petr Skobelev. "Multi-Agent Real Time Scheduling System for Taxi Companies". In: *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2009, pp. 29–36.

[259] Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. "A Taxonomy for Task Allocation Problems with Temporal and Ordering Constraints". In: *Robotics and Autonomous Systems* 90 (2017), pp. 55–70.

[260] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. "Model Checking Continuous-Time Markov Chains by Transient Analysis". In: *Proceedings of the International Conference on Computer Aided Verification (CAV)*. Springer. 2000, pp. 358–372.

[261] Raymond Marie. "Calculating Equilibrium Probabilities For λ (n)/Ck/1/N Queues". In: *Proceedings of the International Symposium on Computer Performance Modelling, Measurement and Evaluation*. 1980, pp. 117–125.

[262] Eric Schneider, Elizabeth I Sklar, Simon Parsons, and A Özgelen. "Auction-Based Task Allocation for Multi-Robot Teams in Dynamic Environments". In: *Proceedings of the Conference Towards Autonomous Robotic Systems*. Springer. 2015, pp. 246–257.

[263] René De Koster, Tho Le-Duc, and Kees Jan Roodbergen. "Design and Control of Warehouse Order Picking: A Literature Review". In: *European Journal of Operational Research* 182.2 (2007), pp. 481–501.

[264] Zvi Drezner and Dawit Zerom. "A Simple and Effective Discretization of a Continuous Random Variable". In: *Communications in Statistics-Simulation and Computation* 45.10 (2016), pp. 3798–3810.

[265] Stanislav Uryasev. "Conditional Value-at-Risk: Optimization Algorithms and Applications". In: *Proceedings of the IEEE/IAFE/INFORMS Conference on Computational Intelligence for Financial Engineering (CIFEr)*. IEEE. 2000, pp. 49–57.

[266] Thomas Keller and Malte Helmert. "Trial-Based Heuristic Tree Search for Finite Horizon MDPs". In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. Vol. 23. 2013, pp. 135–143.

[267] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. "Learning to Act using Real-Time Dynamic Programming". In: *Artificial intelligence* 72.1-2 (1995), pp. 81–138.

[268] Takayuki Osogami and Mor Harchol-Balter. "Closed Form Solutions for Mapping General Distributions to Quasi-Minimal PH Distributions". In: *Performance Evaluation* 63.6 (2006), pp. 524–552.

[269] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22.

Appendices

A

The Floyd-Warshall Algorithm

The Floyd-Warshall algorithm computes the shortest paths between each pair of nodes on a topological map [250], as shown in Alg. A.1.

Algorithm A.1 The Floyd-Warshall Algorithm [250].

Input: Topological map $\mathcal{T}$, edge cost function $c : E \to \mathbb{R}_{>0}$
Output: Shortest path cost matrix **SP**

1: $\mathbf{SP}(v_i, v_j) = \begin{cases} c((v_i, v_j)) & (v_i, v_j) \in E \\ 0 & v_i = v_j \\ \infty & \text{otherwise} \end{cases}$
2: **for** $v_k \in V$ **do**
3: **for** $v_i \in V$ **do**
4: **for** $v_j \in V$ **do**
5: **if** $\mathbf{SP}(v_i, v_k) + \mathbf{SP}(v_k, v_j) < \mathbf{SP}(v_i, v_j)$ **then**
6: $\mathbf{SP}(v_i, v_j) \leftarrow \mathbf{SP}(v_i, v_k) + \mathbf{SP}(v_k, v_j)$
7: **end if**
8: **end for**
9: **end for**
10: **end for**

Floyd-Warshall considers each path between nodes v_i and v_j by iteratively testing whether travelling through node v_k decreases the cost. Floyd-Warshall maintains a $|V| \times |V|$ matrix $\mathbf{SP}$, where $\mathbf{SP}(v_i, v_j)$ stores the shortest path cost between v_i and v_j, and the cost of edge traversal is captured using a cost function c. Floyd-Warshall runs in $\mathcal{O}(|V|^3)$ time, and can be adapted to return the shortest paths in addition to their cost.

B

Labelled Real-Time Dynamic Programming (LRTDP)

Labelled real-time dynamic programming (LRTDP) is a trial-based heuristic tree search method [266] for SSP MDPs that exhibits anytime behaviour, i.e. a good but suboptimal policy is synthesised quickly, and improved with time [87]. LRTDP initialises the value of states using an admissible heuristic, so that states unlikely to contribute to the optimal policy are not explored. Moreover, states are labelled upon convergence, which allows the search to terminate upon finding an optimal policy, unlike real-time dynamic programming [267].

We present LRTDP in Alg. B.1. LRTDP initialises the value function V_π with admissible heuristic h (line 2), and then runs trials until convergence, or upon exceeding a fixed computational budget (lines 3-5). For each trial (lines 8-32), we sample from the initial state $\bar{s}_\mathcal{M}$ of MDP $\mathcal{M}$ under greedy policy π until a goal

Algorithm B.1 LRTDP [37, 87].

Input: MDP $\mathcal{M}$, initial state $\bar{s}_{\mathcal{M}}$, admissible heuristic h, convergence threshold θ
Output: Greedy policy π, value function V_π

```
1: function LRTDP(M, s̄_M, h, θ)
2:     V_π ← h
3:     while s̄_M not labelled solved and time left do
4:         LRTDP-TRIAL(M, s̄_M, θ)
5:     end while
6: end function
7:
8: function LRTDP-TRIAL(M, s̄_M, θ)
9:     visited ← empty stack
10:    s ← s̄_M
11:    while s not labelled solved do
12:        push s onto visited
13:        // Check for SSP goal state
14:        if s ∈ G_M then
15:            break
16:        end if
17:        // Perform Bellman backup & update policy
18:        Q_π(s,a) ← c_M(s,a) + Σ_{s'} δ_M(s,a,s') · V_π(s'), ∀a ∈ A
19:        π(s) ← argmin_{a∈A} Q_π(s,a)
20:        V_π(s) ← Q_π(s, π(s))
21:        // Sample successor MDP state
22:        s' ← sample according to δ_M(s, π(s), s')
23:        s ← s'
24:    end while
25:    // Update solved label for visited states
26:    while visited ≠ empty stack do
27:        s ← pop the top of visited
28:        if ¬CHECK-SOLVED(s, θ) then
29:            break
30:        end if
31:    end while
32: end function
```

state is reached (line 14). Alg. B.1 assumes goal states are reached with probability 1, however LRTDP can be adapted to handle dead ends [249]. A Bellman backup is performed at each state LRTDP visits, which updates the Q values, value function, and policy at that state (lines 18-20). The value function V_π maintains the expected cost to the goal from a state under policy π. The Q value for a state/action pair

Algorithm B.2 `CHECK-SOLVED`, a convergence check for LRTDP [37, 87].

Input: State s, convergence threshold θ
Output: Convergence flag `retVal`

```
 1: function CHECK-SOLVED(s, θ)
 2:     retVal ← True
 3:     open, closed ← empty stack
 4:     push s onto open
 5:     while open ≠ empty stack do
 6:         s ← pop the top of open
 7:         push s onto closed
 8:         // If not converged
 9:         if res(s) > θ then
10:             retVal ← False
11:             continue
12:         end if
13:         a* ← argmin_{a∈A} Q_π(s, a)
14:         for s' ∈ {s'' ∈ S | δ_M(s, a*, s'') > 0} do
15:             if s' not labelled solved and s' ∉ open ∪ closed then
16:                 push s' onto open
17:             end if
18:         end for
19:     end while
20:     // If true, entire subtree has converged
21:     if retVal then
22:         for s ∈ closed do
23:             label s solved
24:         end for
25:     else
26:         // Run Bellman backups back through the subtree
27:         while closed ≠ empty stack do
28:             s ← pop the top of closed
29:             V_π(s) ← min_{a∈A} Q_π(s, a)
30:         end while
31:     end if
32: end function
```

is the immediate action cost, plus the value of subsequently following π to the goal. In lines 26-31, the `CHECK-SOLVED` function is used to update the 'solved' label for states visited during a trial. A solved state does not need to be explored any further. Visiting a solved state terminates the trial (line 11). If the initial state $\bar{s}_{\mathcal{M}}$ is solved, LRTDP terminates with an optimal policy.

In Alg. B.2, we present the `CHECK-SOLVED` function. LRTDP maintains a search tree, which forms the explored portion of the MDP; `CHECK-SOLVED` explores the sub-tree rooted at a state s. If the residual of s, i.e. the change in $V_\pi(s)$ since s last received a Bellman backup, is below a threshold θ (line 9), and all of its descendants in the sub-tree are solved, s is labelled as solved (lines 21- 24). If s is not solved, Bellman backups are performed on the unsolved states in the sub-tree (lines 27-30).

C

Fitting PTDs using Moment Matching

In this appendix, we discuss moment matching techniques for PTDs. An empirical mean μ can be matched with an exponential distribution with rate $\frac{1}{\mu}$, but is insufficient for capturing many distributions. In [261], a Coxian or generalised Erlang is used to match the mean μ and variance σ^2, dependent on the square coefficient of variation $cv^2 = \frac{\sigma^2}{\mu^2}$. If $cv^2 \geq 0.5$, a Coxian with two non-absorbing states is used, i.e. Fig. 2.5d, with $n = 2$.

Definition C.1. When $cv^2 \geq 0.5$, the Coxian that matches μ and σ^2 is a PTD $\mathcal{P} = \langle \{s_1, s_2, s^f\}, init, \Delta, s^f \rangle$, where $init(s_1) = 1$, $\Delta(s_1, s_2) = p_1\lambda_1$, $\Delta(s_1, s^f) = (1 - p_1)\lambda_1$, and $\Delta(s_2, s^f) = \lambda_2$; $p_1 = \frac{1}{2 \cdot cv^2}$, $\lambda_1 = \frac{2}{\mu}$, and $\lambda_2 = \frac{1}{\mu \cdot cv^2}$.

If $cv^2 < 0.5$, the Coxian branching probability $p_1 > 1$, which is invalid. Therefore, a generalised Erlang with n non-absorbing states is used, as in Fig. 2.5c.

Definition C.2. When $cv^2 < 0.5$, the generalised Erlang that matches μ and

σ^2 is a PTD $\mathcal{P} = \langle \{s_1, ..., s_n, s^f\}, init, \Delta, s^f \rangle$, where $init(s_1) = 1$, $\Delta(s_1, s_2) = p\lambda$, $\Delta(s_1, s_f) = (1-p)\lambda$, $\Delta(s_n, s^f) = \lambda$, and $\Delta(s_i, s_{i+1}) = \lambda$, $\forall i \in [\![2, n-1]\!]$. The number of non-absorbing states $n = \lceil \frac{1}{cv^2} \rceil$, the rate $\lambda = \frac{1-p+np}{\mu}$, and the branching probability p is given by:

$$p = 1 - \frac{2n \cdot cv^2 + n - 2 - \sqrt{n^2 + 4 - 4n \cdot cv^2}}{2 \cdot (n-1) \cdot (cv^2 + 1)}. \tag{C.1}$$

If $cv^2 \geq 0.5$, the generalised Erlang branching probability p is invalid, and the Coxian formulation is required. Matching the variance improves the quality of fit against the observed data. This can be improved further by matching the third moment, as in [268], but this increases the PTD size, and thus complexity [52].

D

Fitting PTDs using Maximum Likelihood Estimation

In this appendix, we present the expectation maximisation (EM) algorithm for hyper-Erlangs introduced in [51]. EM algorithms perform MLE by iteratively evaluating the expected log likelihood, and then selecting distribution parameters to maximise it [269]. Hyper-Erlangs are a weighted mixture of M Erlangs, where each Erlang branch $m \in [\![1, M]\!]$ has weight α_m and r_m non-absorbing states, where each transition has rate λ_m (see Fig. 2.5b). The EM algorithm in [51] fixes the number of branches M and non-absorbing states per branch $r_m, \forall m \in [\![1, M]\!]$, and estimates the weights α_m and rates λ_m. Formally, from a set of observations $D = \{d_1, ..., d_K\}$, the parameters $\Theta = [\alpha_1, ..., \alpha_M, \lambda_1, ..., \lambda_M]$ are selected to maximise the log likelihood,

Algorithm D.1 Hyper-Erlang EM Fitting Algorithm [51].

Input: Observations $D = \{d_1, ..., d_K\}$, branch sizes $r_m, \forall m \in [\![1, M]\!]$
Output: Hyper-Erlang parameters $\Theta = [\alpha_1, ..., \alpha_M, \lambda_1, ..., \lambda_M]$
1: **while** not converged **do**
2: **//Compute probability of d_k in each branch using Erlang PDF**
3: $\Pr(d_k \mid \lambda_m) \leftarrow \frac{\lambda_m^{r_m} d_k^{r_m-1} e^{-\lambda_m d_k}}{(r_m-1)!},\ \forall m, k$
4: **//E-step: Compute marginal distribution of latent variable**
5: $\Pr(m \mid d_k, \Theta) \leftarrow \frac{\alpha_m \Pr(d_k|\lambda_m)}{\sum_{i=1}^{M} \alpha_i \Pr(d_k|\lambda_i)},\ \forall m, k$
6: **//M-step: Select parameters to maximise log likelihood**
7: $\alpha_m \leftarrow \frac{1}{K} \sum_{k=1}^{K} \Pr(m \mid d_k, \Theta),\ \forall m$
8: $\lambda_m \leftarrow \frac{r_m \sum_{k=1}^{K} \Pr(m|d_k,\Theta)}{\sum_{k=1}^{K} d_k \Pr(m|d_k,\Theta)},\ \forall m$
9: **end while**

which is expressed as a weighted sum over the hyper-Erlang branches:

$$\max_{\Theta} \log \mathcal{L}(\Theta \mid D) = \sum_{k=1}^{K} \log \left(\sum_{m=1}^{M} \alpha_m \Pr(d_k \mid \lambda_m) \right), \tag{D.1}$$

where $\Pr(d_k \mid \lambda_m)$ is the probability of observing d_k in branch m, given rate λ_m. The logarithm of the weighted sum makes Eq. D.1 difficult to optimise directly [51]. To handle this, the branch m that synthesised observation d_k is considered a latent variable. This is unobservable, however the distribution over branches for d_k can be evaluated with Bayes rule. The log of the sum is removed by conditioning the expected log likelihood on the distribution over branches. We present the hyper-Erlang EM procedure in Alg. D.1.

Alg. D.1 updates the hyper-Erlang parameters until convergence, with respect to the parameters or the log likelihood. In each iteration, the probability $\Pr(d_k \mid \lambda_m)$ of observing d_k in each branch m is computed using the Erlang PDF, which is more efficient than the matrix formulation in Eq. 2.13 (line 3). In the expectation step (line 5), rather than compute the full log likelihood, it is sufficient to compute the distribution over branches for each observation d_k, which is the product of α_m and $\Pr(d_k \mid \lambda_m)$, normalised. In lines 7-8, the parameters are optimised to maximise the log likelihood. The weight for each branch is the average probability an

observation is generated by it. The rate λ_m for each branch is computed by setting the derivative of the log likelihood with respect to λ_m to zero. In Alg. D.1, the hyper-Erlang branch sizes are fixed, however these can be optimised by analysing the quality of fit under different configurations [51].

E

Sequential Single-Item Auctioning (SSI)

SSI auctions [200] allocate a task per round for efficient allocation while considering dependencies between tasks. In Alg. E.1, we present the SSI algorithm for the auctioneer and robots. The auctioneer begins by offering the tasks to all robots (line 2), and then conducts a multi-round auction, where each round allocates a single task. In each round, the auctioneer waits for robots to submit bids, which represent the increase in cost if that robot were allocated the task (line 7). Upon receiving the bids, the auctioneer determines the winner, i.e. the robot with the minimum bid across all tasks, and notifies the robots (lines 10-11). In the next round, only the winner needs to recompute bids, as for standard SSI they remain unchanged for all other robots.

At the start of the auction, robot r_i removes any tasks from its schedule that appear in the set of tasks to be auctioned (line 19). This facilitates online reauctioning during execution. Robots then compute a bid for each task $\tau \in T$.

Algorithm E.1 The SSI algorithm [200, 205].

Input: Robot set R, task set T, partial schedules $\phi_i, \forall i \in [\![1, n]\!]$
Output: Updated schedules $\phi_i, \forall i \in [\![1, n]\!]$

function `AUCTIONEER`(R, T)
    `offerTasks`(R, T)
    `toWait` $\leftarrow R$
    while $T \neq \emptyset$ **do**
        // **Wait for robots to submit bids**
        for $r_i \in$ `toWait` **do**
            $\texttt{bid}_i \leftarrow$ `receiveBids`(r_i)
        end for
        // **Determine and announce winner**
        $r^*, \tau^* \leftarrow \text{argmin}_{i,\tau}\, \texttt{bid}_i^\tau$
        `announceWinner`(r^*, τ^*)
        `toWait` $\leftarrow \{r^*\}$
        $T \leftarrow T \setminus \{\tau^*\}$
    end while
end function

function `ROBOT`(ϕ_i, T)
    `needToBid` $\leftarrow$ `True`
    `removeTasks`(T, ϕ_i)
    while $T \neq \emptyset$ **do**
        if `needToBid` **then**
            for $\tau \in T$ **do**
                // **Compute and send bid**
                $\phi_i^{+\tau} \leftarrow$ `insertIntoSchedule`(ϕ_i, τ)
                $\texttt{bid}_i^\tau \leftarrow SC(\phi_i^{+\tau}) - SC(\phi_i)$
                `sendBid`($\texttt{bid}_i$)
            end for
            `needToBid` $\leftarrow$ `False`
        end if
        $r^*, \tau^* \leftarrow$ `receiveWinner`()
        // **If winner, update schedule**
        if $r^* = r_i$ **then**
            $\phi_i \leftarrow \phi_i^{+\tau}$
            `needToBid` $\leftarrow$ `True`
        end if
        $T \leftarrow T \setminus \{\tau^*\}$
    end while
end function

First, robot r_i inserts τ into a cost-minimising position in their schedule ϕ_i, which can be found by solving a travelling salesperson problem, or using the cheapest insertion heuristic (line 24) [200]. The cost of adding τ into ϕ_i is then evaluated by computing the schedule cost SC before and after τ is added (line 25). If a robot wins a task, it is added to its schedule for the next round of bidding (line 33).

Despite its handling of task dependencies, SSI is greedy and suboptimal. In each round, SSI allocates the task with the lowest bid, without considering how this affects the other unallocated tasks. However, SSI is sufficient for many applications, and is efficient enough to support online reauctioning for solution improvement.