A Learning and Planning Framework for Robust Task Allocation for Heterogeneous Robot Teams

by

Bo Fu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Robotics)
in the University of Michigan
2024

Doctoral Committee:

Professor Kira Barton, Co-Chair
Assistant Professor Maani Ghaffari, Co-Chair
Associate Professor Dmitry Berenson
Professor Benjamin Kuipers
Assistant Professor Vasileios Tzoumas



Bo Fu
bofu@umich.edu
ORCID iD: 0000-0002-2078-3781

© Bo Fu 2024

DEDICATION

To my parents, family, friends, and advisors. My work would not be possible without your support.

ACKNOWLEDGMENTS

I want to express my sincere gratitude to my advisor, Professor Kira Barton, for her guidance and support during my doctoral study. She not only guided me on how to work on top-tier research but also showed me, with her examples, how to think logically and creatively as an educated human being. Her commitment to academic superiority and outstanding guidance helped me develop and mature my skills both technically and personally as a researcher. I would also like to thank Professor Maani Ghaffari Jadidi, who became my co-advisor during the early stage of my Ph.D. study. He generously offered many insightful suggestions for my Ph.D. research, which really benefited my journey. I want to thank Professor Dawn Tilbury for her invaluable suggestions on being a good researcher and her guidance along smart manufacturing-related projects. I want to thank all other members of my committee - Professor Benjamin Kuipers, Professor Dmitry Berenson, and Professor Vasileios Tzoumas. Their insightful comments, dedicated time, mentorship, direction, and conversations about personal goals have been immensely beneficial and highly valued. I want to thank the Robotics team - Denise Edmund, Kimberly Mann, David Parkin, Dan Newman, and Damen Provost for always being so helpful, patient, and supportive during student activities and research.

Next, I would like to express my gratitude to every member of Barton Research Group and Computational Autonomy and Robotics Laboratory (CURLY). I am profoundly thankful for the stimulating exchange of ideas, joint efforts, and constructive debates with colleagues like Mingjie Bi, Tyler Toner, Tong Chen, Tribhi Kathuria, Abigail Rafter, Joseph Wilson, Michael Quann, Ilya Kovalenko, and Laxmi Prasad Poudel, among others. Their input has significantly enhanced my comprehension of the subject and encouraged me to investigate new viewpoints. I want to thank Michelle Ji, Shreya Phirke, Aarish Shah, Nolan Theeke-Gallego, and Xiang Gao for helping me develop experimental and simulation platforms for my research. I want to thank Loubna Baroudi, Max Toothman, Max Wu, Nazanin Farjam, Dory Yang, Yaqing Xu, Zahra Afkhami, Joaquin Gabaldon, Ding Zhang, Isaac Spiegel, Efe Balta, Changbai Tan, Leo Tse, Yassine Qamsane, Paola Rioja, Ali Bahrami, Angelo Hawa, Keith Ng, Kaifan Yue, Patrick Linford, Aleksandra Dudek, Gabriel Antoniak, Minghan Zhu, Ray Zhang, Chien Erh (Cynthia) Lin, Tzu-Yuan (Justin) Lin, Sangli Teng, Junzhe Wu, Wenzhe Tong, Bruce JK Huang, and Lu Gan. They enriched my daily life and nurtured a sense of belonging in the lab.

Beyond my research groups, I am immensely grateful to my friends and peers at the University of Michigan, such as Professor Min Deng, Professor Xi Wang, Zhixi Chen, Hongrui Yu, Hang Song, Dr. Kaiwen Liu, Chao Chen, Yifei Shao, Haochen Wu, and others, for their constant support in my academic endeavors and personal well-being.

Finally, I owe a profound debt of gratitude to my parents for their relentless love and encouragement. Their steadfast faith in my potential, coupled with the sacrifices they have made to further my education, have been the foundation of my resolve. I am eternally thankful for their guidance and the principles they have ingrained in me, which have been instrumental in molding my character and driving my progress.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGMENTS iii

LIST OF FIGURES viii

LIST OF TABLES x

LIST OF ACRONYMS xi

ABSTRACT xiii

CHAPTER

1 Introduction 1

1.1 Motivation 1
1.2 Objectives and Contributions 4
1.3 Dissertation Outline 9
1.4 Open-source Software 9

2 Robust Task Assignment, Scheduling, and Routing for Heterogeneous Robot Teams . 10

2.1 Introduction 11
2.1.1 Contributions 15
2.1.2 Concepts and Terminology 16
2.2 Related Work 17
2.3 Routing, Scheduling, and Risk Minimization 20
2.3.1 Heterogeneous Teaming Problem Description 21
2.3.2 Routing Model 23
2.3.3 Risk Minimization Model - A Mixed-Integer Program 24
2.3.4 Sample Average Approximation (SAA) 29
2.3.5 The L-shaped Algorithm 30
2.4 Decompose Agent Flows into Paths 33
2.4.1 Minimum Energy Cost Rounding 34
2.4.2 Minimum Max-Energy Flow Split 36
2.5 Experiments and Results 37
2.5.1 Decompose Agent Flows into Paths 38
2.5.2 Capture the Flag 40

2.5.3 Robotic Services during a Pandemic . . . 44
2.6 Conclusions and Future Work . . . 52
2.7 Appendix: Non-trivialness of the Flow Decomposition . . . 56
2.8 Appendix: Proof of the Integer-valued Solutions . . . 56

3 Human-robot Matching and Routing for Multi-robot Tour Guiding under Time Uncertainty . . . 60

3.1 Introduction . . . 61
3.2 Related Work . . . 63
3.3 Human-robot Matching and Tour Routing . . . 65
3.3.1 Problem Description and Graphical Model . . . 66
3.3.2 A Mixed-integer Bilinear Program . . . 67
3.3.3 Uncertain Travel and Visiting Time . . . 71
3.3.4 A Branch-and-Cut Approach for an Exact Solution . . . 73
3.3.5 A Large Neighborhood Search Approach . . . 73
3.4 Local Navigation and multi-robot Simulation . . . 76
3.4.1 Local Navigation Planner . . . 77
3.4.2 Habitat-AI Simulation Environment . . . 77
3.4.3 Uncertainties in Navigation . . . 77
3.5 Experiments and Results . . . 80
3.5.1 Computational Evaluation of the Tour Planning . . . 80
3.5.2 Multi-robot Simulation in Habitat-AI . . . 84
3.6 Conclusions and Future Work . . . 88

4 Learning Task Requirements and Agent Capabilities for Multi-agent Task Allocation 90

4.1 Introduction . . . 91
4.2 Related Work . . . 94
4.2.1 Optimization model-based Multi-agent Task Allocation . . . 94
4.2.2 Learning Optimization Models from Data . . . 94
4.3 Learning Task Requirements and Agent Capabilities as Linear Constraints . . . 95
4.3.1 Task Performance Function . . . 95
4.3.2 Task Requirements and Agent Capabilities . . . 96
4.3.3 Learning the Task Requirements and Agent Capabilities . . . 97
4.3.4 Motivation and Effect of the Assumptions . . . 100
4.4 Task Allocation with Learned Requirements and Agent Capabilities . . . 101
4.4.1 Graphical Model . . . 101
4.4.2 A Mixed-integer Program for Task Allocation . . . 102
4.5 Computational Evaluation . . . 105
4.5.1 Test Cases . . . 105
4.5.2 Accuracy and Computational Cost . . . 106
4.6 Simulation Example . . . 108
4.6.1 Learning the Capability Model . . . 109
4.6.2 Task Allocation with the Learned Capabilities . . . 112
4.7 Conclusions and Future Work . . . 113
4.7.1 Conclusions . . . 113

4.7.2 Limitations and Future Work . . . 114

5 Learning-based Intra-team Trajectory Planners for Tightly Cooperative Tasks 115

5.1 Introduction . . . 116
5.2 Related Work . . . 118
5.2.1 Optimization-based Methods for Tightly Cooperative Multi-agent Tasks . 118
5.2.2 Learning-based Methods for Tightly Cooperative Multi-agent Tasks . . . 119
5.3 Factorized Learning for GNN-based MARL . . . 121
5.3.1 Problem Description . . . 121
5.3.2 Global Reward and Joint Policy Learning . . . 122
5.3.3 Composite Individual Reward and Factorized Learning . . . 126
5.3.4 Composite Individual Reward and GNN-based Learning . . . 130
5.3.5 Comparison Between Network Structures . . . 132
5.4 Simulation Results . . . 133
5.5 Conclusions and Future Work . . . 135
5.5.1 Conclusions . . . 135
5.5.2 Future Work . . . 136

6 Conclusion and Future Directions . . . 138

6.1 Contributions . . . 139
6.1.1 Multi-agent Task Allocation for Large-scale Complex Tasks . . . 139
6.1.2 Task Requirement and Agent Capability Learning . . . 139
6.1.3 Intra-team Trajectory Planners for Tightly Cooperative Tasks . . . 140
6.2 Broader Impact . . . 141
6.2.1 Comprehensive Hierarchical Multi-agent Framework . . . 141
6.2.2 Heterogeneous and Generalizable Multi-agent System . . . 142
6.2.3 Robust and Adaptive Multi-agent System . . . 142
6.3 Future Directions . . . 143
6.3.1 Distributed Multi-agent Task Allocation . . . 143
6.3.2 Real-time Replanning for Task Allocation . . . 144
6.3.3 Intra-team Hierarchical Task and Motion Planner . . . 146

BIBLIOGRAPHY . . . 147

LIST OF FIGURES

FIGURE

1.1 System architecture and the four research contributions 5
1.2 Three components of a complex task allocation problem 6

2.1 Graphical model for an example task allocation problem 12
2.2 System architecture of the task allocation framework 13
2.3 Graphical illustration of the conditional value at risk 19
2.4 Graphical model with $n_v = 3$ agent species and $n_m = 2$ tasks. 23
2.5 Decompose an agent flow into paths 35
2.6 Graphical model for the minimum max-energy flow split problem 37
2.7 Flow rounding results 39
2.8 Capture the flag setup 40
2.9 Comparison between the task assignment performance in the capture-the-flag game . . 44
2.10 The average number of agents assigned to attack in the capture-the-flag game 45
2.11 The success rate of the task allocation models 50
2.12 The optimality gap of the task allocation models 51
2.13 The mean probability of success for the tasks using the three models. 52
2.14 The energy consumption and the task completion probability of the task allocation models 52
2.15 The relative approximation gap due to sample average approximation 53
2.16 Trade-off between energy cost and the probability of task completion 53
2.17 Routes for the robotic services. 54
2.18 Round the agent flow 57
2.19 Split the agent flow 58
2.20 Totally unimodularity of the rounding problem. 59

3.1 Framework of the multi-robot tour guiding 63
3.2 Bipartite matching problem 64
3.3 Vehicle routing problem 64
3.4 A graphical model for the tour guiding problem 67
3.5 Key notations in a simultaneous matching and routing problem 68
3.6 Top view of one simulation environment for the tour guiding simulation. 78
3.7 Example views of the robots conducting tour guiding tasks 78
3.8 Planned and executed paths in a tour guiding task 79
3.9 Trade-off between the tour time limit and dropped requests ratio 82
3.10 Influence of the estimated time uncertainty on dropped requests 84
3.11 Distribution of the POIs and the routes of the robots in an example tour guiding problem 85

3.12 Actual tour time of the robots when the environmental uncertainty changes 86
3.13 Planned tour of the robots assuming under execution uncertainty 87
3.14 Actual tour time of the robots when the estimated uncertainty changes 88
3.15 Planned tour of the robots, assuming no action uncertainty 89

4.1 Multi-agent exploration and block picking 92
4.2 System architecture of the learning framework 93
4.3 Capability-based task requirement constraints 97
4.4 Finding the tightest approximation of the task requirement boundary 98
4.5 Graphical model for optimization-based task allocation 102
4.6 Prediction errors compared to the baseline 108
4.7 Screenshot of the multi-agent simulation environment 109

5.1 Table lifting task 122
5.2 A comparison of the policy network structures and the rewards 123
5.3 Network structure for joint policy learning 124
5.4 Network structure for factorized learning 127
5.5 Network structure for GNN-based reinforcement learning 131
5.6 Table lifting task: cumulative reward. 134
5.7 Table lifting task: lifted tables. 135
5.8 Table lifting task: achieved goals. 136

LIST OF TABLES

TABLE

2.1 A comparison between STRATA and CTAS. 19
2.2 Definition of the notation for the heterogeneous teaming problem. 24
2.3 The computational cost of the flow decomposition. 39
2.4 Capability distributions of the four agent species in the capture-the-flag game. 42
2.5 General task descriptions of the capture-the-flag game. 42
2.6 Task requirements specified as capability distributions. 42
2.7 Definitions of the capability types in the robotic service example. 46
2.8 Stochastic agent capabilities in the robotic service example. 47
2.9 Stochastic task requirements in the robotic service example. 47
2.10 Task descriptions in the robotic service example. 48
2.11 Three teaming models in the robotic service example. 48
2.12 A comparison of the results of the three teaming models. 55
2.13 Team configurations for the robotic services given by the three models. 55

3.1 Definition of the notations in a simultaneous matching and routing problem. 69
3.2 Discrete action space for the robot. 77
3.3 Algorithm types tested for the simultaneous matching and routing problem. 81
3.4 Ratio of the dropped requests according to the tour guiding planner. 81
3.5 The planned tour and teams guided by the robots. 84

4.1 Definition of the notations. 103
4.2 Test cases of different sizes and the training times 106
4.3 Qualitative agent capabilities for the exploration and block picking task. 109
4.4 Qualitative task requirements for the exploration and block picking task. 110
4.5 Task completion time and threshold selection for the tasks. 111
4.6 Learned agent capabilities for the exploration and block picking task. 112
4.7 Learned task requirements for the exploration and block picking task. 112
4.8 Task teams according to the learned task requirements. 113

5.1 Table-lifting task: input states and output actions for joint policy learning setup. 125
5.2 Table-lifting task: inputs and outputs for factorized and GNN-based learning. 129
5.3 A comparison of the network structures. 132

LIST OF ACRONYMS

VRP Vehicle Routing Problem

CVRP Capacitated Vehicle Routing Problem

SVRP Stochastic Vehicle Routing Problem

CVaR Conditional Value at Risk

CTAS Capability-based robust Task Assignment and Scheduling

STRATA Stochastic TRAit-based Task Assignment

HTP Heterogeneous Teaming Problem

CD Complex Task Dependencies

ST Single-Task Robot

MR Multi-Robot Task

TA Time-extended Assignment

IA Instantaneous Assignments

SLAM Simultaneous Localization and Mapping

MDP Markov Decision Processes

MIP Mixed-integer Program

MILP Mixed-integer Linear Program

ILP Integer Linear Program

LP Linear Program

CPU Central Processing Unit

GPU Graphical Processing Unit

TU Totally Unimodular

SMRP Simultaneous Matching and Routing Problem

POI Places of Interest

LNS Large Neighborhood Search

SNS Small Neighborhood Search

ROS Robot Operating System

NN Neural Network

GNN Graph Neural Network

RNN Recurrent Neural Network

RL Reinforcement Learning

MARL Multi-agent Reinforcement Learning

PBT Population-based Training

PPO Proximal Policy Optimization

MLP Multi-layer Perception

PDDL Planning Domain Definition Languages

TAMP Task and Motion Planning

LLM Large Language Models

ABSTRACT

Recent progress in sensing and control technologies has significantly expanded the deployment of robotic systems in increasingly complex scenarios. These scenarios often feature more intricate tasks, larger operational scales, and environments marked by higher levels of uncertainty. As these evolving challenges exceed the capabilities of individual robots, the necessity for multi-agent robotic systems has become more pronounced, highlighting their importance in tackling such complex and uncertain situations. On the other hand, there is still a lack of coordination frameworks that leverage the full capability of the single agents during complex multi-agent tasks.

This dissertation tries to bridge the gap between single and multi-agent tasks through the development of a hierarchical multi-agent feedback planning system. This framework contains several key components: a task allocation planner responsible for forming agent teams and planning routes and schedules; an intra-team trajectory planner that ensures coordinated movements among agents for tightly coupled tasks; and a learning model that estimates agent capabilities and task requirements to provide feedback for updating the plans.

The task allocation planner is developed based on stochastic programming. It simultaneously optimizes task decomposition, task assignment, task schedule, and agent routes, enabling the systems to generate optimal solutions for complex task allocation scenarios where all these problems are coupled. The planner leverages functional heterogeneity within the agent teams to handle complex tasks, explores parallel execution to improve task efficiency, and considers different types of uncertainties to generate robust plans. Experiments show that the framework can scale to hundreds of agents and tasks, significantly advancing over existing frameworks.

The learning model is proposed to estimate and update the model parameters required by the task allocation planner, including task requirements and agent capabilities. Based on a linear program, the learning model can estimate task requirements and agent capabilities for a problem of ten thousand data samples within seconds. This efficient learning process enables real-time updates, and therefore, the model can serve as feedback for the task allocation planner to close the loop for online re-planning.

For tightly cooperative multi-agent tasks, even when the task has been decomposed into single-agent roles and assigned to each agent, the trajectories of the agents can still intersect or interfere. A graph neural network-based multi-agent reinforcement learning framework is proposed to generate

collision-free, collaborative trajectories for the multi-agent system. This innovative approach is designed to encode task constraints among the agents, efficiently train a policy network, enable distributed communication between agents, and enhance the scalability of the trajectory planning.

We show the practicality and generalizability of the feedback planning system by applying it to different case studies, including capture-the-flag games, multi-robot pandemic robotic services, multi-robot tour guidance, multi-robot exploration and block picking, and multi-robot manipulation. To validate the scalability of our system, we rigorously evaluate the proposed optimization and learning algorithms across a range of problem sizes and configurations. This comprehensive assessment confirms that the system can adapt and perform effectively in diverse and increasingly complex settings.

The framework enhances the knowledge of integrating the multiple levels of a multi-agent system in one consistent framework. While the application in this dissertation lies in multi-robot cooperation, the framework, along with the insights derived from it, holds significant potential to impact other fields, such as healthcare, urban planning, agriculture, financial decision-making, and collaboration in smart manufacturing.

CHAPTER 1

Introduction

1.1 Motivation

Advances in robotic sensing and control have enabled robotic applications to have an ever-growing scope [1–9]. On the other hand, the increasingly diverse and complex nature of robotic tasks soon surpasses the capability of a single robot: limited by design rules and actuator/sensor power, even the most competent robot cannot independently manage all real-world tasks. For example, in a large-scale automated warehouse, there can be hundreds of storage and retrieval requests each minute [10]. A single robot is not able to fulfill all the requests in time. As another example, in a robotic construction task, some structures are only possible to be built when multiple parts are simultaneously assembled [11]. A single manipulator is not able to hold multiple parts at the same time. In robotic exploration (e.g., the DARPA subterranean challenge[1] and Mars exploration), the area to explore is usually so large that using a single robot to fulfill the task would surpass the time limit. In addition, the different geographic conditions within the area require heterogeneous types of mobility that do not exist in a single agent. Researchers also reveal that multi-robot localization and tracking lead to higher accuracy than using a single robot [12].

This trend fosters the recent proliferation of multi-agent system in real-world applications, including robotic agriculture [13], warehouse management [14], construction [15], defense [16], exploration [17–21], and surveillance [22–25]. The advantages of multi-agent systems include the

[1] https://www.darpa.mil/program/darpa-subterranean-challenge

robustness to agent failures, resilience of team configuration, and lower maintenance costs (a large robot with the same task capabilities is usually harder to design and costlier due to the system complexity) [26].

However, transitioning from a single robot to a multi-agent system involves more than just operating multiple individual robots and their algorithms in one area. This scaling up presents numerous challenges. Consider the DARPA subterranean challenge as an example. In this scenario, each robot is capable of performing simultaneous localization and mapping independently. As a team, these robots aim to share information to create a global map of the subterranean area. This task necessitates additional algorithmic components for inter-agent communication, map merging and updating, as well as map storage and retrieval due to the increased scale. Furthermore, if the team comprises heterogeneous agents, additional planning and scheduling frameworks become essential for assigning the appropriate agent to the correct unexplored region.

Suppose every agent has its perception and planning system setup and can conduct multiple individual actions, roles, or single-agent tasks. One major gap between multi-agent tasks and single-agent tasks is the task allocation: if every single agent knows when and where to conduct a specific single-agent action, then a multi-agent task can be decomposed into such roles, assigned to the individual agents with a coordinating schedule and completed by leveraging existing single-robot perception, planning, and control systems for the specific tasks.

In tightly cooperative tasks, even if each agent knows which action to perform at a specific time, there can still be dependencies between the agent actions due to collision or the need for collaboration. In such a scenario, another gap between multi-agent tasks and single-agent tasks is an intra-team trajectory planner that resolves conflicts or considers the collaboration constraints for dependent agent actions.

A real-world multi-agent application usually involves the operation of multiple heterogeneous agents (robots, vehicles) to complete multiple heterogeneous sub-tasks in unstructured environments. Such environments lead to uncertainties in energy cost, time, agent states, and task progress. To introduce autonomy into and maximize the performance in such applications, the task allocation

framework should inevitably model heterogeneity and uncertainty.

Heterogeneity can be classified into the structural and functional categories [27]. Structural heterogeneity usually leads to less phenomenal differences that can be described as parameters, e.g., maximum velocity or energy capacity. Functional heterogeneity describes distinct sensor and actuator capabilities of the robots, usually leading to fundamental differences between task capabilities, e.g., the ability to fly, pick up objects, or remove debris. Most existing research on multi-agent systems has focused on teams of either homogeneous agents or agents with structural heterogeneity. For instance, vehicle routing problems (VRP) try to find the optimal routes for a fleet of homogeneous vehicles to visit a set of task targets. Capacitated vehicle routing problems (CVRP) additionally model one structural heterogeneity of the agents, assuming vehicles have a limited carrying capacity for the goods that must be delivered. Among the few previous works that deal with functional heterogeneity in a complex task setting, most optimization models are designed for use with a specific application. Therefore, there is a need for fundamental methods that provide a more systematic representation for functional heterogeneity that can be generalized within a larger scope of complex tasks.

The concept of uncertainty and disturbances is crucial when dealing with dynamic systems. Two strategies can be adopted to counter failures [28]. First, robust (proactive) planning algorithms that consider the uncertainty pre-execution and generate a plan that is prepared to withstand the uncertain environment. Second, adaptive (reactive) algorithms that recover the system from failure due to uncertainties in real-time. In many situations, the two methods can be combined to improve the robustness of the system further. For instance, considering the effect of uncertainty during pre-execution and selecting a recoverable plan with low cost will facilitate a real-time adaptive process. In a robust task allocation framework, both strategies should be applied and combined to ensure task completion.

Few previous works consider the functional heterogeneity and uncertainty handling of task allocation (task decomposition, role assignment, and scheduling) within a generalizable and scalable framework. This dissertation seeks to develop a framework to bridge the gaps of task allocation

and intra-team trajectory planning between single-agent capabilities and multi-agent task execution. The following challenges should be addressed:

1) develop a generalizable and scalable multi-agent task allocation framework to generate task roles, schedules, and routes for heterogeneous agents;

2) model environmental uncertainties and the uncertainties in agent capabilities and task requirements to improve the robustness of the plans;

3) build a trajectory planner for heterogeneous agents to enable tightly cooperative tasks where the action of one agent affects the action of another agent;

4) create a learning model to estimate and update the model parameters and serve as feedback for the task allocation.

1.2 Objectives and Contributions

This dissertation will develop a feedback planning system for multi-agent systems to generate robust teaming plans and update the plans according to the feedback. The system should be able to generate team-level task allocation and scheduling plans, create intra-team level trajectory plans for tightly cooperative tasks where agent trajectories are dependent on each other, and estimate the agent capabilities and task requirements to inform the update of the plans. The system diagram of the feedback planning framework is shown in Fig. 1.1. It contains three main components.

C1: Multi-agent Task Allocation for Complex Tasks and Heterogeneous Agents

As shown in Fig. 1.2, the fundamental problems [29] that arise when applying a functionally heterogeneous multi-agent system to a mission containing multiple complex tasks include understanding how to decompose a task into elemental tasks (how), determining which agents should be assigned to a particular elemental task (who), and deriving a schedule that enables the heterogeneous team to successfully complete the task (when). Therefore, considering the whole problem as task allocation; then, task allocation = task decomposition + assignment + scheduling. A task

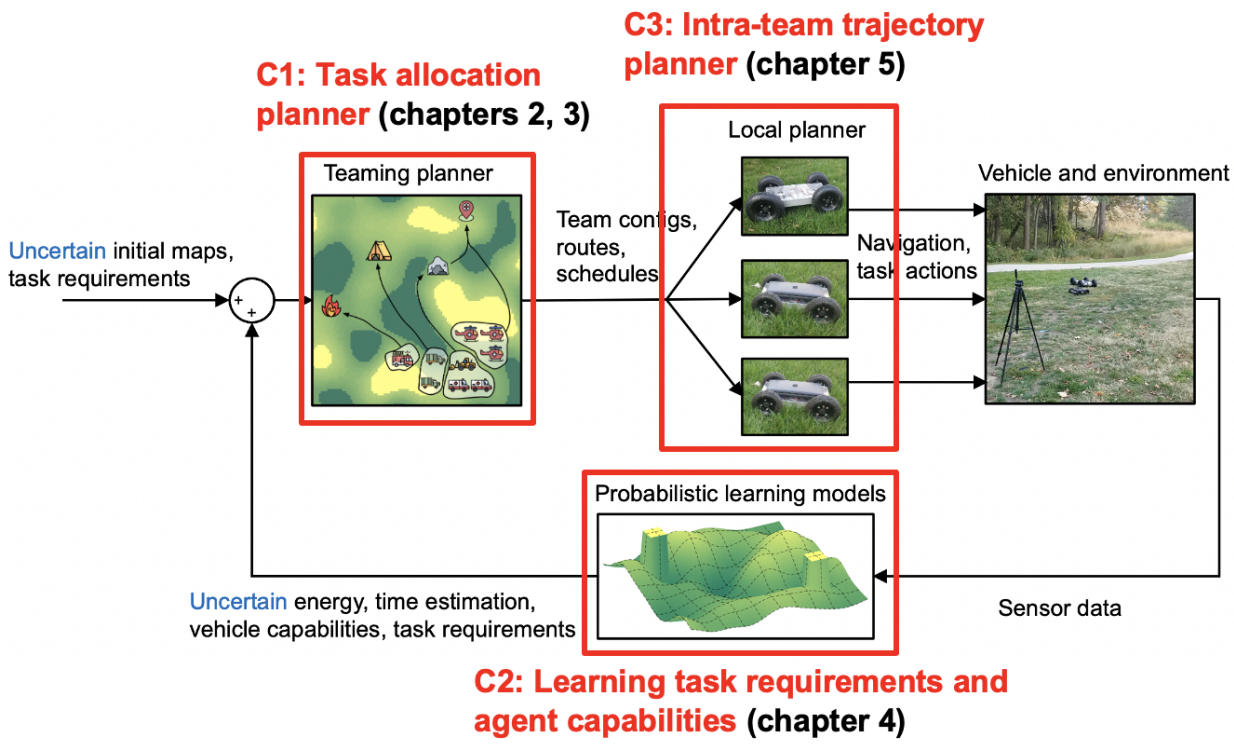


Figure 1.1: The system architecture and the three research contributions of the proposed work. C1: a task allocation framework that generates schedules and teams for complex tasks. C2: a learning model that estimates time cost, energy cost, agent capabilities, and task requirements. C3: a distributed intra-team planner that generates trajectories for heterogeneous robot cooperation.

is considered complex if there exist multiple decompositions and it is unknown which decomposition should be selected without simultaneously considering task assignment and schedule planning (i.e., the problem is coupled) [29]. Few multi-agent task allocation frameworks for complex tasks are both generalizable to different tasks and scalable to the problem size. A fundamental research question to answer is: what is the correct level of abstraction in task allocation and the corresponding model to achieve a balance among optimality, scalability, and generalizability?

Uncertainties in the understanding of the environment and inaccurate estimation of the task requirements and agent capabilities can degrade the task performance or result in task failure. A robust task allocation plan should secure task success under such uncertainty by providing redundant capabilities in the team. Mathematically, a more robust task assignment plan means there is a higher probability that a team's capabilities satisfy task requirements. However, such a redun-

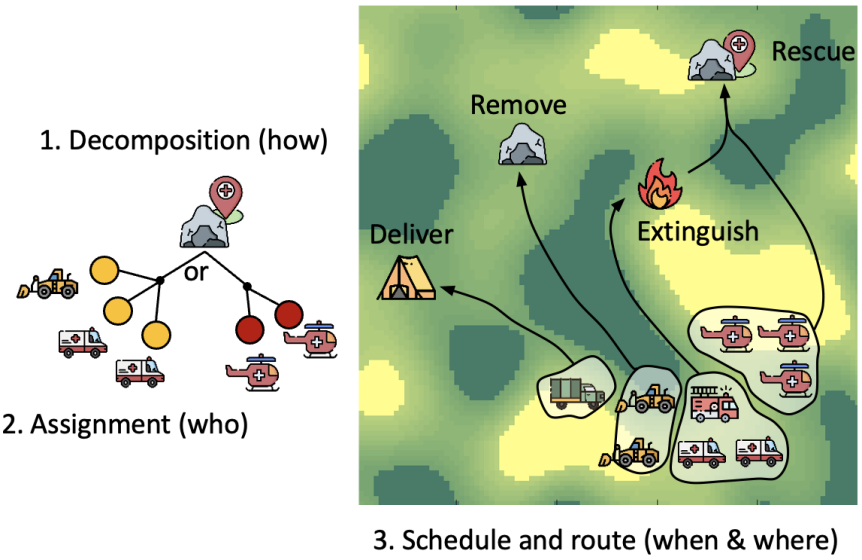


Figure 1.2: A complex task allocation problem that contains the task decomposition, assignment, and scheduling and routing components.

dancy increases the time and energy cost, and a careful trade-off is required through optimization. Therefore, the associated research question is: how to ensure robustness when there is uncertainty in heterogeneous capabilities and tasks?

To answer these research questions, we develop a task allocation framework that generates routes, schedules, assignments, and teams for complex tasks satisfying:

- The task and agent representation model generalizes to heterogeneous practical problems.
- The planning process scales to problems consisting of hundreds of agents.
- The environmental uncertainties and variations in the task requirements and agent capabilities are incorporated into the math model to trade off between task completion and energy/time usage.

Contribution 1 is addressed in Chapters 2 and 3.

C2: Learning and Updating the Task Requirements and Agent Capabilities

Optimization model-based frameworks are usually generalizable to different real-world scenarios and can provide an optimality bound. However, most existing optimization frameworks for

multi-agent task allocation assume the modeling parameters are given. However, many task-related modeling parameters are hard to determine in practice, even for an expert. Therefore, learning the parameters for model-based becomes an important supplement to the optimization model-based task allocation frameworks.

While standard methods exist for estimating time cost, energy cost, and optimal paths between two locations, the estimation algorithm for the agent capability and task requirement model used in our framework has to be established. Existing work [30] has accomplished to learn task requirements given agent capabilities. However, one research question/gap is: Is there a way to simultaneously learn agent capabilities and task requirements? The task allocation framework described in **C1** contains many modeling parameters that should be estimated according to historical data. In addition, the closed-loop replanning in Fig. 1.1 requires the modeling parameters to be updated according to task states and sensor data during the task execution.

We develop a learning model to estimate the energy, time, and task requirement model for the developed optimization framework.

Chapter 4 is dedicated to fulfilling contribution 2.

C3: Intra-team Trajectory Planners for Tightly Cooperative Tasks

One major gap between multi and single-agent tasks is task allocation: once the tasks are allocated to the correct agent teams, single-agent controllers can be leveraged to handle the single-agent tasks. However, it is assumed that the cooperation between the agents within the team is loose, and the action trajectory of one agent does not affect/depend on another agent. This assumption is correct for applications such as formation control of aerial vehicles, where each agent needs to fly its own planned path, or multi-agent delivery systems in large open spaces [31], where the collision/cooperation between the agents is not considered. However, for more tightly coupled cooperative tasks like bimanual or multi-arm manipulation [11, 32, 33], handover behaviors [34], joint weight lifting [35], and exploration using a joint global map [36], one agent's action affects another agent's. In such cases, knowing the task allocation and schedule of the agents and tasks still does not decouple the multi-agent tasks into conductible single-agent plans.

An intra-team multi-agent trajectory planner for cooperative sub-tasks is required in such cases. The sub-tasks considered here are short periods of highly interactive tasks where one agent's trajectory-level actions depend on another agent. The research question, given all possible methods for trajectory planning, is: how to plan the trajectories for tightly cooperative tasks to respect the dependencies between agent actions but also consider variable team size and heterogeneous agents and tasks?

There are many related works for trajectory-level cooperation between homogeneous agents. From a loose cooperation to a tight cooperation, researchers have investigated topics in the formation control of swarm robots [37], multi-agent pathfinding with collision avoidance in a crowded environment [38, 39], robotic construction using swarms [15], multi-agent surveillance and object tracking [22–24, 40], and cooperative manipulation (including bimanual manipulation) [11, 32–34, 41–43]. Note that for formation control and multi-agent pathfinding, the system performances are improved when trajectory-level interactions are explicitly modeled. Most optimization-based frameworks are task-specific because a specific model is needed for each type of collaboration. Most of the learning-based frameworks concatenate the agent states and treat the system as a single agent. Such a fully centralized framework limits its scalability.

With the advancement of reinforcement learning theory and simulation, there is a proliferation of multi-agent reinforcement learning applications. However, developing individual reward functions and network structures to efficiently handle the constraints of a tightly cooperative task remains an open question.

We develop a scalable distributed intra-team planner that generates trajectories for robots in tightly cooperative cooperation. The framework is evaluated using cooperated manipulation tasks and shows superior training efficiency and flexibility in team sizes.

Contribution 3 is addressed in Chapter 5.

1.3 Dissertation Outline

The rest of this dissertation is organized as follows. Chapter 2 details a multi-agent task allocation planner for complex tasks to generate teaming plans, routes, and agent schedules. The uncertainties in the energy consumption, task requirements, and agent capabilities are considered in the optimization. Chapter 3 illustrates the application of a multi-agent task allocation planner in a multi-robot tour guiding problem where human requests are considered. The effects of the time uncertainty are modeled and evaluated. Chapter 4 formulates a learning framework to estimate agent capabilities and task requirements for multi-robot task allocation, which serves as feedback to the task allocation planner. Chapter 5 develops a reinforcement learning-based intra-team trajectory planner for tightly cooperative tasks. Chapter 6 concludes this dissertation and Chapter 7 provides ideas for future work based on current limitations.

1.4 Open-source Software

This section provides a list of open-source packages released with this dissertation.

- A stochastic programming-based task allocation planner for multi-agent systems.
  https://gitlab.com/barton-research-group/open/resilient_team_planner
- A model that learns agent capabilities and task requirements for multi-agent task allocation.
  https://gitlab.com/barton-research-group/open/learn-multiagent-taskreq
- A Gazebo and ROS-based simulator for heterogeneous multi-agent exploration and block pick tasks.
  https://gitlab.com/barton-research-group/open/teaming_simulation
- A simultaneous matching and routing solver for a multi-robot tour guide problem.
  https://github.com/UMich-CURLY/tourguide_routing_matching

CHAPTER 2

Robust Task Assignment, Scheduling, and Routing for Heterogeneous Robot Teams

This chapter tries to address the contribution 1 in Fig. 1.1. We develop a stochastic programming framework for multi-agent task allocation where task decomposition, assignment, and scheduling problems are simultaneously optimized. The framework can be applied to heterogeneous mobile robot teams with distributed sub-tasks. Examples include pandemic robotic service coordination, explore and rescue, and delivery systems with heterogeneous vehicles. Due to their inherent flexibility and robustness, multi-agent systems are applied in a growing range of real-world problems that involve heterogeneous tasks and uncertain information. Most previous works assume one fixed way to decompose a task into roles that can later be assigned to the agents. This assumption is not valid for a complex task where the roles can vary and multiple decomposition structures exist. Meanwhile, it is unclear how uncertainties in task requirements and agent capabilities can be systematically quantified and optimized under a multi-agent system setting. A representation for complex tasks is proposed: agent capabilities are represented as a vector of random distributions, and task requirements are verified by a generalizable binary function. The conditional value at risk (CVaR) is chosen as a metric in the objective function to generate robust plans. An efficient algorithm is described to solve the model, and the whole framework is evaluated in two different practical test cases: capture-the-flag and robotic service coordination during a pandemic (e.g., COVID-19). Results demonstrate that the framework is generalizable, scalable up to 140 agents and 40 tasks for the example test cases, and provides low-cost plans that ensure a

high probability of success.

The remainder of this chapter is organized as follows. Sec. 2.1 motivates the fundamental problems in a complex task allocation problem and explains the importance of modeling the uncertainty in dynamic systems. Sec. 2.2 briefly introduces the related work of multi-agent task allocation and concludes with research gaps that this work investigates. Sec. 2.3 describes the general problem mathematically and presents a risk minimization framework. Sec. 2.3.5 introduces an algorithm to solve the risk minimization problem and output a general routing plan described as network flows of agent movements. Sec. 2.4 proposes a network flow decomposition problem to generate routing plans and schedules for individual agents. The two-step process in Sections 2.3.5 and 2.4 improves the scalability of the framework compared to a solution method that outputs individual plans directly within one optimization. Sec. 2.5 discusses the experiments and results. Finally, Sec. 2.6 concludes the chapter and provides ideas for future work based on current limitations.

This chapter is adapted from our paper [44, 45].

Supplementary videos and open-source code for this project can be found at:
https://brg.engin.umich.edu/publications/robust-task-scheduling.

2.1 Introduction

The escalation in demand for complex and large-scale robotic tasks has given rise to new challenges that single robotic arms are insufficient to manage. The need for multi-robot systems has become increasingly clear. The importance of developing multi-agent systems lies in their enhanced task capability, robustness, and resilience.

The reason that a team of less powerful robots can achieve the same or more complex tasks with the above advantages is that the functional heterogeneity within the team, i.e., distinct sensor and actuator capabilities of the robots, can complement each other during a task. In contrast to structural heterogeneity (e.g., maximum velocity or energy capacity), functional heterogeneity usually leads to fundamental differences between task capabilities (e.g., the ability to fly or transport peo-

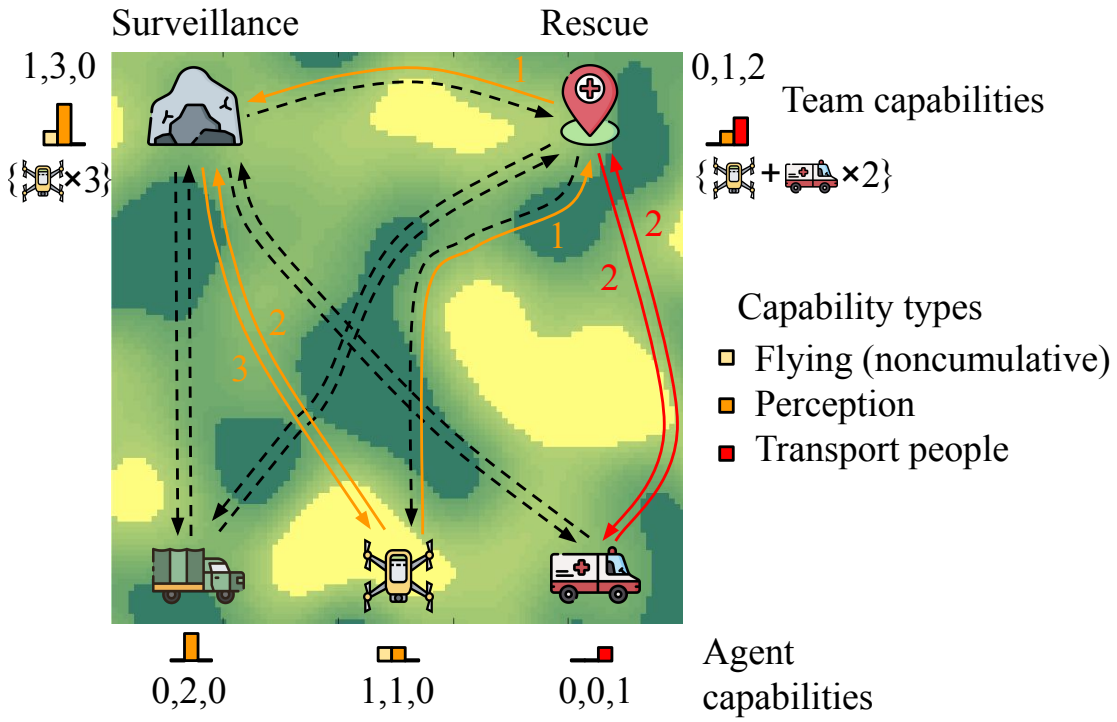


Figure 2.1: A graphical model for a problem with three agent species and two tasks. There are three types of capabilities: flying, perception, and transport people. The capability values are defined according to actual tasks. For instance, a quadcopter that can fly, explore an area at the rate of 1 m^2/s on average, and seat 0 people, has the expected capability values as $[1, 1, 0]^T$. The rescue task requires the ability of perception and transporting people. The surveillance task requires both flying and perception capabilities. The capability to fly is noncumulative, which is defined in in Sec. 2.1.2 and equation (2.2). The framework will consider time, energy, agent capabilities, and task constraints, and generate the schedules and routes for the teams. In this example, there are three agent species for selection and after the optimization, the rescue task is assigned to a team of two ambulances and a quadcopter, while the surveillance task is conducted by three quadcopters.

ple) [27]. The concepts of agent and team capabilities and task requirements are described in Sec. 2.1.2.

As mentioned in Sec. 1, the fundamental problems [29] that arise when applying a functionally heterogeneous multi-agent system to a mission containing multiple complex tasks include: understanding how to decompose a task into elemental tasks (how), determining which agents should be assigned to a particular elemental task (who), and deriving a schedule that enables the heterogeneous team to successfully complete the task (when). Therefore, considering the whole problem as task allocation; then, task allocation = task decomposition + assignment + scheduling. The con-

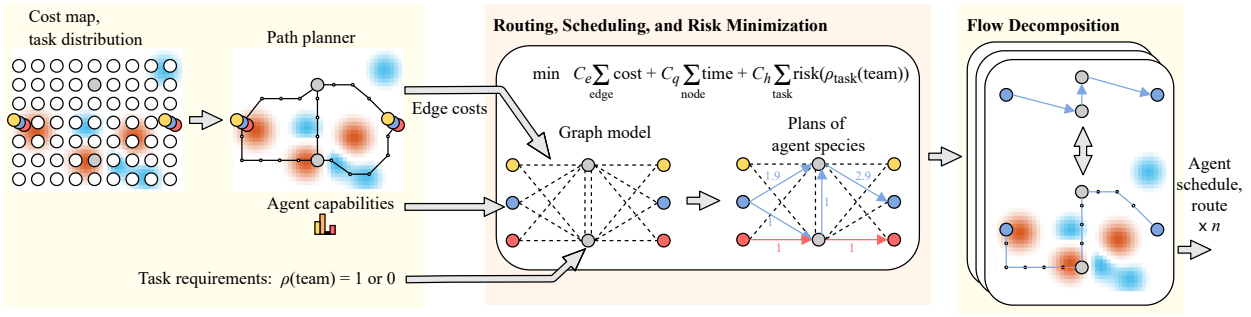


Figure 2.2: System architecture. Input: cost (energy) maps (an example is in Sec. 2.5.3), agent capabilities, and task requirements. Output: agent routes, schedules, and team formations. There are two major components. A routing, scheduling, and risk minimization model (Sec. 2.3) generates a network flow for each agent species. And then these networks are further decomposed into individual agent plans through a flow decomposition model (Sec. 2.4).

cepts of task decomposition, elemental tasks, and complex tasks are formally defined in Sec. 2.1.2. In the example in Fig. 2.1, the task decomposition problem considers how to decompose a task's required capabilities into agent roles such that after being assigned to the appropriate agents, the team capability surpasses the requirements. Task scheduling determines the routes such that the agents will arrive at a same task together while minimizing the energy and time usage.

A task is considered complex if there exist multiple decompositions and it is unknown which decomposition should be selected without simultaneously considering task assignment and schedule planning (i.e., the problem is coupled) [29].

Among the few previous works that deal with functional heterogeneity in a complex task setting, most optimization models are designed for use with a specific application. Therefore, there is a need for fundamental methods that provide a more systematic representation that can generalize within a larger scope of complex tasks.

An important aspect to consider when dealing with dynamic systems is the concept of uncertainty. Two strategies can be adopted to counter failures [28]: 1) robust planning algorithms that consider the uncertainty pre-execution and generate a plan that is prepared to withstand the uncertain environment; 2) adaptive and reactive algorithms that recover the system from failure due to uncertainties in real-time.

In a task allocation setting, one source of the uncertainty is the inaccurate estimation of or the inherent variation within the task requirements and agent capabilities. We apply the robust planning strategy to generate more reliable plans in the pre-execution stage. A robust task allocation plan and schedule should secure task success under such uncertainty by providing redundant capabilities in the team. Mathematically, a more robust task assignment plan means there is a higher probability that the number of capabilities in a team exceeds the number required by the task. However, such a redundancy increases the time and energy cost, and a careful trade-off is required. In Sec. 2.3, we will introduce how we encode this robustness in the optimization and determine a trade-off.

In this chapter, we present a **Capability-based robust Task Assignment and Scheduling (CTAS)** framework to optimize task decomposition, assignment, and scheduling simultaneously within a stochastic task model, which can generalize to multiple practical scenarios. Consider a set of heterogeneous agents (can be robots/vehicles/humans) and complex tasks; the proposed framework will form agent teams, schedules, and routes to minimize energy and time costs for completing the task combined with the risk of non-completion. We define this class of problems as the **heterogeneous teaming problem (HTP)**. The system architecture is summarized in Fig. 2.2 and will be discussed in the following sections.

Note that the system focuses on the optimization that distributes the capabilities to the tasks at a low cost. The information about agent capabilities, task requirements, and spatial traveling costs are gathered and a routing and teaming plan is generated premission in a centralized planner. Individual plans are sent to the agents and do not change during the task execution. Though the plans do not change, they are designed to withstand the uncertainty and variations in capability estimation and are less likely to fail. Future extensions will consider a recovery strategy and update the plans during the task execution. Communication between agents that enables information gathering and plan synchronizing is not the focus of this chapter and will be considered in future work.

2.1.1 Contributions

In [44], we dealt with a deterministic variation of such a problem, where we assumed exact information (instead of a distribution) of the agent capabilities and task requirements was known. In this work, we develop a generalizable framework for task assignment and scheduling that systematically represents heterogeneous and uncertain task requirements and agent capabilities. This work provides the following contributions

1) The development of a modeling framework that captures uncertainties within the task requirements and agent capabilities.

2) The derivation of a cost function that incorporates the concept of risk within the minimization.

3) Reformulation of the heterogeneous teaming problem (HTP) to provide a more scalable algorithm that is solved by using a flow decomposition subproblem.

4) The implementation of a capture-the-flag game simulation to compare the task assignment performance to a baseline algorithm.

The problem size depends on the number of tasks, agent species, and agents per species. With the reformulation in this work, the routing of individual agents is decoupled and postponed to a flow decomposition subproblem which reduces the size of the optimization program for the HTP. The HTP only considers the behavior of an entire agent species, and its size no longer depends on the number of agents per species. The complexity of the framework is still exponential, as we use mixed-integer programming to find the exact optimal solution [46]. But the reformulation suppresses the growth such that the framework can solve practical problems with up to 140 agents and 40 tasks.

The open-source code is available at:
https://brg.engin.umich.edu/publications/robust-task-scheduling

2.1.2 Concepts and Terminology

In this section, we provide definitions of multiple concepts that will be used throughout the chapter. For some of the terminology, we refer to [29], but simplify the description to provide only the core meaning.

Agent capabilities. A numeric vector that describes the agent's fitness to conduct a task.

Team capabilities. A set of fitness values aggregated from agent capabilities in the team.

Cumulative capabilities. A capability that sums the agent capabilities across an entire team to get the team capability.

Noncumulative capabilities. A capability that does not sum. In this chapter, noncumulative team capability is the minimum of the agent capabilities in the team.[1]

Task requirement. A set of constraints that relate the task and agents. Usually, it is specified as a number of required agents, agent actions, or agent capabilities that have to be satisfied. In this chapter, it is defined as a set of required team capabilities and will be described in Sec. 2.3.

Task decomposition. Representing a task as a set of elemental tasks (there can be constraints between these subtasks), such that completing the subtasks completes the original task.

Task assignment. Identifying which specific agent should handle an elemental task.

Elemental task: A task that is not decomposable and should be assigned to only one agent.

Compound task. A task that can be decomposed into a set of elemental tasks in only one fixed way.

Complex task. A task that can be decomposed into elemental tasks in more than one way, i.e., following different decompositions, the resulting sets will contain different numbers or types of elemental tasks.

[1] In Fig. 2.1, the capability to fly is noncumulative: the team can fly only when all agents in the team can fly.

2.2 Related Work

According to the taxonomy in [29, 47–51], our framework for the HTP deals with a task allocation problem in the category CD-[ST-MR-TA]: complex task dependencies (CD), single-task robot (ST), multi-robot tasks (MR), and time-extended assignment (TA). It is one of the most challenging task assignment categories and has been considered in few previous works in literature. The time-extended assignment considers scheduling, in contrast to instantaneous assignments (IA), which only make matches between tasks and robots.

While many previous works about multi-robot tasks are in the category [ST-MR-TA], most of their papers do not deal with complex tasks. Their papers decompose a multi-robot task into elemental tasks that can be assigned to a subset of single agents. If such a decomposition exists, the task decomposition problem is presolved and decoupled from the assignment and scheduling problem. Many previous works in vehicle routing [27, 52], job shop scheduling [53–55], and robotic soccer games [56, 57] deal with such non-complex (or compound) tasks. We refer interested readers to the survey paper [29] for more work with compound tasks.

There are systems dealing with complex tasks, but their models do not generalize and are applied to one specific problem. Examples include fire extinguishing and debris removal [58, 59], and a coverage problem of unmanned aerial/ground vehicles with recharging behaviors [23], where tasks are considered complex because multiple decomposition combinations are possible in the problem space.

Some frameworks for CD-[ST-MR-TA] created in previous work generalize to a type of problem, but most of them have scalability issues and are only applicable to a small group of robots. For instance, Bayesian network representations consisting of tasks, observations, constraints, and action nodes can be used to represent a task with interdependent elemental subtasks [60, 61]. Hierarchical tree networks (HTN) are used in [62–65] to represent tasks where leaf nodes are roles for assignment. Planning domain definition language (PDDL) can be used to represent a task as a graph of states (nodes) and actions (edges) [66–74]. Temporal planning can be considered an extension to the basic PDDL [75–79]. In addition to a graph of predefined actions and state-space

representation, temporal planning allows continuous temporal constraints and interdependence between subtasks to be encoded as predefined logical formulas. However, these representations based on actions and states are usually applied to systems with less than five agents. The number of states in their trees/graphs could explode rapidly with respect to the number of agents and available actions. The underlying solver (e.g., graph search algorithms) would take a long time to explore a subset of the high-dimensional space. Such a scalability issue limits its application to larger multi-agent systems.

Uncertainty exists widely in the estimation of task requirements and agent capabilities. Some recent works capture such uncertainty explicitly in their models. However, the metrics they optimize do not necessarily result in a high-probability task requirement satisfaction (which is considered robust). For example, the work of [80] models the agent action uncertainty in Markov decision processes (MDP) and optimizes the expected objective. In [56, 57, 81], an agent's capabilities are represented as a Gaussian random vector where the uncertainty is captured in the distribution. The work of [81] then penalizes the variance of the assigned agent capabilities to limit the uncertainty. However, expectation or variance are not well-justified quantitative metrics in such an optimization problem and can result in problematic solutions. For instance, if the required capability of a task is matched precisely, it is reasonable to limit the variances within a small threshold. However, if the team's capability surpasses the task requirement by a lot, larger variances are acceptable. As another example, optimizing an expected cost (average performance) can be problematic for safety-critical applications. In [82], the authors directly maximize the probability that enough capabilities are assigned. However, probability is non-convex in the decision variables, and global optima are hard to obtain for large problem cases.

Our proposed model considers a generalizable framework for allocating agent capabilities for complex tasks under capability and requirements uncertainty, and it can be applied to systems consisting of hundreds of agents. We represent the agent capabilities as a vector of random distributions (not necessarily Gaussian) and the task requirements within a function of the team capabilities (defined in Sec. 2.3). The task requirement is verified once the aggregated capabilities of

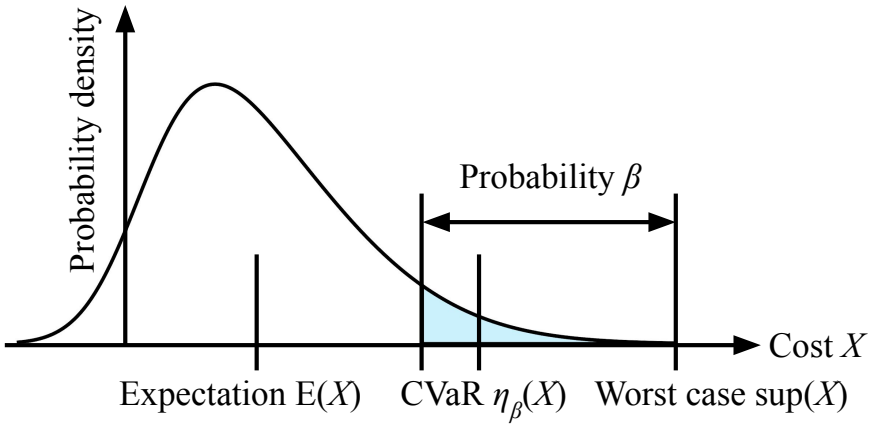


Figure 2.3: Graphical illustration of CVaR, defined as expected cost of the worst β-proportion of the cases. Denoted as $\eta_\beta(\cdot)$, it is a function of the random distribution with β as a hyper-parameter.

Table 2.1: A comparison between STRATA and CTAS.

	STRATA[81]	CTAS (Ours)
Agent capabilities	continuous/Gaussian	continuous/stochastic
Capability types	cumulative	cumulative/noncumulative
Task requirements	and	and/or
Scheduling	no	yes
Uncertainty control	limit variance	minimize CVaR
Optimization type	continuous nonlinear	mixed-integer nonlinear

the team drive the binary function to one. We then solve the task decomposition, assignment, and scheduling problem simultaneously, where we optimize the time, energy, and robustness of task allocation. As mentioned in the introduction, a robust allocation should ensure the capability of the team exceeds the task requirement with a high probability. We choose to minimize the conditional value at risk (CVaR) of (requirement $-$ capability), which is consistent[2] with maximizing the probability $P(\text{capability} \geq \text{requirement})$ [83]. CVaR is a provably sensible measurement of the uncertainties in practical applications [84]. It is widely accepted by the finance community and appears with a growing frequency in recent robotic applications for risk-aware single-robot control [85–88] and multi-robot coordination [28, 89]. The definition of CVaR is illustrated in Fig. 2.3.

Our problem can be considered a coalition formation with spatial and temporal constraints.

[2]According to [83], maximizing the probability is equivalent to minimizing the Value at Risk, and the Conditional Value at Risk is a risk metric with additional math properties that facilitate optimization problems.

Multiple models have been proposed in previous work [90–92]. The work of [91] has a similar application to ours, but it assumes that there exists a predefined utility function that outputs a value for a team configuration. While an abstract function generalizes the problem space, explicitly defining the function values for all team configurations could be hard in practice. We provide our task requirement and agent capability model to avoid such an explicit definition. The work of [93] and its extension [30] use a similar requirement and capability model. However, they do not consider the uncertainty in their capability model. A graph search-based algorithm is applied in their task allocation process, limiting the scalability of the number of agents (the size of the graph).

STRATA [81, 94] shares many similarities with our proposed approach, CTAS, including stochastic agent capability vectors and task requirements on the team's capabilities. Therefore, we choose STRATA as our baseline algorithm in one of the experiments in Sec. 2.5. However, STRATA falls in the area of CD-[ST-MR-IA], where it assumes the tasks happen at the same time, which simplifies the scheduling problem. Meanwhile, our task model, represented using requirement functions, is more expressive. TABLE 2.1 provides a more detailed comparison between the two models. The meaning of 'and/or' in the table is defined in Sec. 2.3. Note that the definition of 'noncumulative' in STRATA differs from ours. STRATA claims that it can deal with noncumulative capability types (such as the speed of an agent). However, it thresholds on a value and then treats the binary value after thresholding as a cumulative capability. For noncumulative types, we define the team's capability to be the minimum of all agents and require instead that all agents in the team meet the minimum task requirement. STRATA is not able to enforce such requirements for their noncumulative capabilities.

2.3 Routing, Scheduling, and Risk Minimization

In this section, we formally define the heterogeneous teaming problem with uncertain agent capabilities and task requirements. We describe the task requirement functions and agent capability vectors used to represent the task structure, organize them in a graphical model, and then encode

the task allocation problem as a stochastic mixed-integer program whose objective jointly consists of time, energy, and risk cost.

The modeling assumptions are listed as follows:

- Tasks are distributed spatially and can be completed after a known time once a team with the required capabilities arrives.
- The agent capabilities can be modeled as a vector of known continuous random distributions.
- The task requirement can be verified by a binary function of the agent team's capabilities. The parameters in the function are task-required capabilities and are known continuous random distributions.
- Different types of capabilities are independent and do not affect each other.
- The completion of one task does not depend on other tasks. But the scheduling dependency between tasks is considered, i.e., agents should arrive at the same time for a task, and the delay of one task could delay the subsequent tasks. The model can be extended with other time constraints, e.g., precedence constraints.

2.3.1 Heterogeneous Teaming Problem Description

We simplify the graphical model in [44] for this work. Consider a set of agent species $V = \{1, \cdots, n_v\}$, capability types $A = \{1, \cdots, n_a\}$, and tasks $M = \{1, \cdots, n_m\}$. Note that $V, A, M \subset \mathbb{N}$ are three integer sets. Each agent species $k \in V$ is associated with a non-negative capability vector $\mathbf{c}_k = [c_{k1}, \cdots, c_{kn_a}]^\mathsf{T}$, where c_{ka} is a random variable with a known distribution, representing the uncertain task capability of type $a \in A$ for agent $k \in V$. Each task $i \in M$ requires an agent team with appropriate capabilities that drives a task requirement function $\rho_i(\cdot)$ to 1.

A task requirement function is a binary function of a similar structure as (2.1). The logical operators $\geq$, $\wedge$, and $\vee$ are 'greater than or equal to', 'and', and 'or' that return 1 if their conditions are satisfied, and 0 otherwise. γ_{1a} describes the task 1's requirement on capability a, and is modeled

as a random variable with a known distribution. We define $\gamma_{1a} = 0$ if it does not appear in (2.1). Note that (2.1) is an example of a task that requires four types of capabilities. In practice, there can be an arbitrary number of $\wedge$ and $\vee$, theoretically.

$$\rho_1(\alpha_1, \alpha_2, \cdots) = [(\alpha_1 \geq \gamma_{11}) \vee (\alpha_2 \geq \gamma_{12})] \wedge [\alpha_3 \geq \gamma_{13}] \wedge [\alpha_4 \geq \gamma_{14}]. \tag{2.1}$$

$$\alpha_a = \begin{cases} \sum_{k \in V} c_{ka} \cdot y_{k1}, \ a \text{ is cumulative} \\ \min_{k \in V} c_{ka} \cdot r_{k1}, a \text{ is noncumulative} \end{cases}, \forall a \in A. \tag{2.2}$$

α_a is the capability a of the agent team. y_{ki} is the number of agents at task $i \in M$ with species $k \in V$. r_{ki} is binary and indicates whether there is an agent with species $k \in V$ at task $i \in M$. Depending on whether the capability is cumulative, we can compute α_a according to (2.2). An example of noncumulative capabilities is the speed limit of a team. It equals the speed of the slowest moving agent in the team.

The requirement to drive these functions to 1 could be satisfied with appropriate team formation planning. This requirement constraint can be encoded as linear constraints according to [44]. Note that the only part that could introduce non-convexity is the logic $\vee$, which takes the union of two feasible regions. Reducing the number of $\vee$ operators is desired as it facilitates the optimization.

With stochasticity in task requirements and agent capabilities, the goal is to determine the optimal task schedule for a selected set of agents, such that the energy, time, and path constraints are satisfied, and the energy cost and risk of the task's non-completion are jointly minimized.

The task is considered complex for two reasons. First, the logic $\vee$ operator results in multiple decompositions. Secondly, there is no fixed way to decompose the requirements in (2.1) into single-agent elemental tasks; instead, they are optimized together with the assignment and scheduling problem.

There are inter-schedule dependencies between tasks since the overall time cost for one agent to achieve one task depends on the schedule of other agents. For example, if one agent delays its

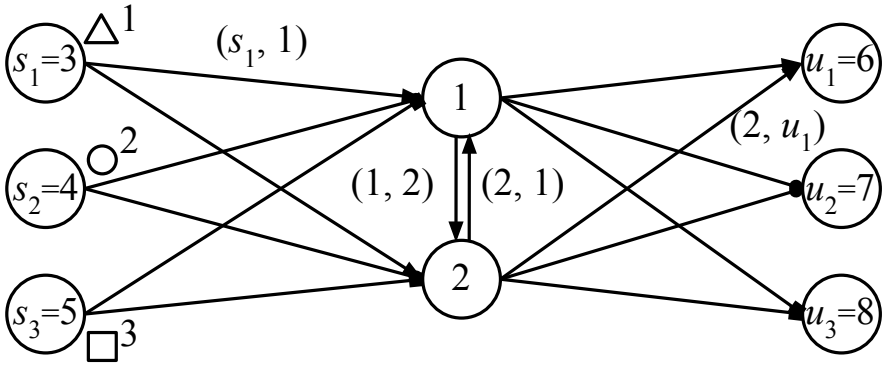


Figure 2.4: Graphical model with $n_v = 3$ agent species and $n_m = 2$ tasks.

work at another task and arrives late at the current task, the time cost increases for all agents at the current task.

2.3.2 Routing Model

As shown in Fig. 2.4, we first define a directed graph $G = (N, E)$, with a set of vertices $N = S \cup U \cup M$ and edges E. $M = \{1, \cdots, n_m\}$ is the set of task nodes. $S = \{n_m + 1, \cdots, n_m + n_v\}$ and $U = \{n_m + n_v + 1, \cdots, n_m + 2n_v\}$ are the sets of start and terminal nodes for each agent species $k \in V$. The prefix n_m and $n_m + n_v$ are added such that the vertex indices in M, S, and U are unique integers. The size of the vertices set M, S, and U are n_m, n_v, and n_v, respectively. Note that these nodes can represent the same or different physical locations in the real world. A practical example of this graphical model is in Fig. 2.1.

The edge set $E = \{(i, j) | \, \forall i \in S \cup M, \; j \in M \cup U\}$: there are edges from the starts to all tasks, from all tasks to the terminals, and between all the tasks. The edges are associated with their time and energy costs and the pre-computed real-world paths between two node locations. There is an edge between every task node pair in Fig. 2.4, but in practice, we only add an edge if there is a feasible path between two nodes.

Under this setting, the agents of species $k \in V$ should start at s_k, follow the edges to visit a subset of task nodes (often together with other agents), and terminate at u_k. As a result, the agent numbers on the edges form a network flow from the start to the terminal nodes in graph G.

Table 2.2: Definition of the notation for the heterogeneous teaming problem.

	Meaning
x_{kij}	The number of agents on edge (i, j) with species $k \in V$, where node $i, j \in N$. For simplicity, x_{ks_ki} and x_{kiu_k} $\forall k \in V, i \in N$ are abbreviated to x_{ksi} and x_{kiu} since there is no ambiguity. The colored numbers in Fig. 2.1 are solutions to x_{kij} for that example problem.
y_{ki}	The number of agents at task $i \in M$ with species $k \in V$.
r_{kij}	= 1, if $x_{kij} \geq 1$, otherwise 0. (A helper variable indicating whether there are agents with species k on that edge.)
r_{ki}	= 1, if $y_{ki} \geq 1$, otherwise 0. (A helper variable indicating whether there are agents with species k at that task.)
q_i	The time that task $i \in M$ begins.
g_{ki}	The maximum cumulative energy that an agent of species $k \in V$ has spent at node i. (A helper variable.)
b_{kij}	The deterministic energy cost for agent $k \in V$ to travel edge $(i, j) \in E$.
t_{kij}	The deterministic time for $k \in V$ to travel edge $(i, j) \in E$.
t_{ki}	The deterministic time for agent species $k \in V$ to complete its part for task $i \in M$.
C_q	Time penalty coefficient.
C_{large}	A large constant number for the MIP.
B_k	The energy capacity of agent $k \in V$.
B_{large}	A large constant energy for the MIP.
h_i	The conditional value at risk from task $i \in M$.
n_k	The number of agents with species $k \in V$.

2.3.3 Risk Minimization Model - A Mixed-Integer Program

We discuss an MIP model that generates a schedule and agent team for each task and the corresponding agent flow by minimizing the energy cost and the risk of the task's non-completion. Here we provide common notations that will be used in TABLE 2.2. Note that the decision variables are x_{kij}, y_{ki}, r_{kij}, r_{ki}, q_i, and g_{ki}.

Compared to [44], the x_{kij} and y_{ki} variables in this work are defined for agent species instead of individual agents, and their values are now real instead of binary. This change reduces the number of variables and, therefore, decreases the computational cost of the planner. However, as the model does not specify variables for individual agents, the mixed-integer program (MIP) outputs an agent flow of each species instead of path plans for individual agents. We will discuss algorithms for

post-processing the agent flows to get routes for individual agents in Sec. 2.4.

2.3.3.1 Variable Bounds

$$
\begin{aligned}
&x_{kij} \geq 0,\ y_{ki} \geq 0, \quad \forall i, j \in N, \forall k \in V. \\
&r_{kij},\ r_{ki} \in \{0, 1\}, \quad \forall i, j \in N, \forall k \in V. \\
&q_i \geq 0, \quad \forall i \in M \cup U. \quad q_i = 0, \quad \forall i \in S. \\
&g_{ki} \geq 0, \quad \forall i \in M \cup U. \quad g_{ki} = 0, \quad \forall i \in S, \forall k \in V.
\end{aligned}
\tag{2.3}
$$

The numbers of agents from species $k \in V$ on an edge and at a task node are positive and real. Their helper variables are binary. The time and cumulative energy are nonnegative at all nodes and zero at the start nodes.

2.3.3.2 Helper-variable Constraints

$$
\begin{aligned}
&x_{kij} \geq r_{kij}, \quad x_{kij} \leq n_k \cdot r_{kij}, \quad \forall i, j \in N, \forall k \in V. \\
&y_{ki} \geq r_{ki}, \quad y_{ki} \leq n_k \cdot r_{ki}, \quad \forall i \in N, \forall k \in V.
\end{aligned}
\tag{2.4}
$$

Equation (2.4) encodes the relationship between $\{x_{kij},\ y_{ki}\}$ and their helper variables $\{r_{kij},\ r_{ki}\}$.

2.3.3.3 Network Flow Constraints

$$\sum_{i \in S \cup M} x_{kim} = \sum_{j \in U \cup M} x_{kmj}, \quad \forall m \in M, \forall k \in V. \tag{2.5}$$

$$\sum_{i \in M} x_{ksi} \leq n_k, \qquad \forall k \in V. \tag{2.6}$$

$$y_{kj} \leq \sum_{i \in M} x_{ksi}, \quad \forall j \in M, \forall k \in V. \tag{2.7}$$

$$y_{kj} = \sum_{i \in S \cup M} x_{kij}, \quad \forall j \in M, \forall k \in V. \tag{2.8}$$

Equation (2.5)-(2.7) are flow constraints that ensure the agent numbers are smaller than the upper bound, and that the incoming agent number at a node equals the outgoing number. Constraint (2.8) reflects the relationship that the number of agents at a node equals the sum of the agent flows from all incoming edges.

2.3.3.4 Energy Constraints

$$g_{ki} - g_{kj} + b_{kij} \leq B_{\text{large}}(1 - r_{kij}), \forall i, j \in N, \forall k \in V. \tag{2.9}$$

$$g_{ki} \leq B_k, \quad \forall i \in N, \forall k \in V. \tag{2.10}$$

Equation (2.9) ensures that g_{ki} is the cumulative energy of the agent species - if an edge is traveled, the energy b_{kij} is added. And because g_{ki} is the maximum cumulative energy of agent species k at node i, equation (2.10) ensures the energy cost for an agent of species k does not exceed its energy capacity B_k.

These energy constraints form sufficient conditions as they assure the most costly path in the flow network of species k is within the capacity limit B_k. However, it is possible that the most costly path is not picked during the flow decomposition procedure (Sec. 2.4). This is a compromise

made by us along the process of improving the scalability of the model: we replace the variables for individual agents with variables for an entire agent species. With the original variables for individual agents, necessary and sufficient energy constraints were easily imposed.

2.3.3.5 Time Constraints

$$q_i - q_j + t_{kij} + t_{ki} \leq C_{\text{large}}(1 - r_{kij}), \forall i, j \in N, \forall k \in V. \tag{2.11}$$

Equation (2.11) is a scheduling constraint: for an agent, the time duration between two consecutive tasks should be larger than the service time at the previous task plus the travel time. This constraint gives all the agents in the team enough time to arrive before the current task starts.

With the task time variables q_i, the optimization problem can be extended with other user-defined time constraints. For instance, precedence constraints between tasks or time window constraints within which a task should complete [95, 96].

2.3.3.6 Task Requirement Constraints

$$1 = \rho_i(\sum_{k \in V} E(c_{k1}) y_{ki}, \sum_{k \in V} E(c_{k2}) y_{ki}, \cdots), \ \forall i \in M. \tag{2.12}$$

$$1 = \rho_i(\min_{k \in V} E(c_{k1}) r_{ki}, \min_{k \in V} E(c_{k2}) r_{ki}, \cdots), \ \forall i \in M. \tag{2.13}$$

For cumulative capabilities we add constraint (2.12), and for noncumulative capabilities we add constraint (2.13). $E(\cdot)$ is the expectation operator. c_{ki} is the capability value defined in Sec. 2.3.1. Note that an example of $\rho_i(\cdot)$ is in (2.1), and γ_{ia} $(\forall i \in M, a \in A)$ should be replaced with $E(\gamma_{ia})$ here. Though the task requirements and the agents' task capabilities are stochastic, we can add this deterministic constraint that requires the teams' expected capabilities to satisfy the task

requirement by driving the requirement function to 1.

2.3.3.7 Objective Function

$$\min C_e \sum_{k \in V} \sum_{i,j \in N} b_{kij} \cdot x_{kij} + C_q \sum_{i \in U} q_i + C_h \sum_{i \in M} h_i \tag{2.14}$$

$$h_i = \sum_{a \in A \text{ s.t. } \gamma_{ia} \neq 0} h_{ia}, \qquad \forall i \in M, \tag{2.15}$$

$$h_{ia} = \max_{k \in V \text{ s.t. } r_{ki}=1} \eta_\beta \left(-c_{ka} + \gamma_{ia} \right), \quad a \in A, \forall i \in M, \tag{2.16}$$

$$h_{ia} = \eta_\beta (-\sum_{k \in V} c_{ka} \cdot y_{ki} + \gamma_{ia}), \qquad a \in A, \forall i \in M. \tag{2.17}$$

In the objective function (2.14), we want to minimize a weighted combination of the energy cost, task time, and the conditional value at risk of the task's non-completion, where C_e, C_q, and C_h are user-defined weights. For the risk of a task, h_i in (2.15), take task 1 and its example requirement function in (2.1) as an instance. According to (2.1) and (2.15), $h_1 = h_{11} + h_{12} + h_{13} + h_{14}$. To explain this, task 1 requires capabilities 1-4. Assuming the requirements on these capabilities are independently placed, the total risk is the sum of the risks from each requirement.

Consider h_{ia}, the risk on a single capability $a \in A$. According to the example requirement function (2.1), a task requests $\alpha_a \geq \gamma_{ia}$. Both α_a and γ_{ia} are stochastic. To maximize the probability $P(\alpha_a \geq \gamma_{ia})$, we instead minimize the CVaR of $\gamma_{ia} - \alpha_a$. Let the function $\eta_\beta(\cdot)$ be the CVaR of a random variable with probability level β. Then, h_{ia} can be computed according to (2.17) or (2.16) depending on whether capability $a \in A$ is cumulative or not.

Given the uncertainty in the capabilities, the objective function (2.14), together with the deterministic task requirement constraint (2.12), tries to maximize the probability of task success at a low energy and time cost.

2.3.4 Sample Average Approximation (SAA)

Solving the optimization specified by (2.3)-(2.17) requires dealing with stochasticity and non-linearity. In this section, we show how to convert this stochastic mixed-integer nonlinear program (MINLP) to a deterministic mixed-integer linear program (MILP).

Notice that $E(c_{ka})$ in (2.12) and $\eta_\beta(-c_{ka}+\gamma_{ia})$ in (2.16) do not involve decision variables, these can be computed prior to the optimization, given the distribution of c_{ka} and γ_{ia} ($k \in V, a \in A$). The stochasticity is eliminated by the expectation and risk function. The deterministic task requirements in (2.12) can then be represented with a set of linear constraints according to [44]. Constraint (2.16) can also be represented as a linear constraint

$$h_{ia} \geq \eta_\beta\left(-c_{ka}+\gamma_{ia}\right)\cdot r_{ki}, \quad \forall k \in V, a \in A, \forall i \in M. \tag{2.18}$$

The only non-linearity is in (2.17) due to the function $\eta_\beta(\cdot)$ and the decision variable y_{ki}. However, we can linearize (2.17) using the sample average approximation algorithm. Suppose we can represent the random distribution c_{ka} and γ_{ia} by samples $c_{ka}^{(\xi)}$ and $\gamma_{ia}^{(\xi)}$ ($\xi = 1, \cdots, n_\xi$), respectively. Then we can approximate (2.17) with a linear equation (2.19) according to [97]. The approximation converges at the rate of $\mathcal{O}(n_\xi^{-1/2})$ [98]. In the following equation, λ_{ia} is a continuous helper variable for calculating h_{ia} (task $i \in M$ and capability type $a \in A$).

$$\begin{aligned} h_{ia} =& \lambda_{ia} + 1/n_\xi(1-\beta)\cdot \\ & \sum_{\xi=1}^{n_\xi}\left[-\sum_{k\in V} c_{ka}^{(\xi)}\cdot y_{ki} + \gamma_{ia}^{(\xi)} - \lambda_{ia}\right]^+ . \end{aligned} \tag{2.19}$$

$$[x]^+ = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} . \tag{2.20}$$

Finally, the piece-wise linear function in (2.19) can be represented as linear constraints in (2.21).

Note that we add a series of continuous helper variables $w_{ia}^{(\xi)}$ to represent the $[\cdot]^+$ function in (2.20).

$$
\begin{aligned}
h_{ia} &= \lambda_{ia} + \frac{1}{n_\xi(1-\beta)} \sum_{\xi=1}^{n_\xi} w_{ia}^{(\xi)}, \\
w_{ia}^{(\xi)} &\geq -\sum_{k \in V} c_{ka}^{(\xi)} \cdot y_{ki} + \gamma_{ia}^{(\xi)} - \lambda_{ia}, \\
w_{ia}^{(\xi)} &\geq 0.
\end{aligned}
\tag{2.21}
$$

2.3.5 The L-shaped Algorithm

In the previous section, we formulate the stochastic heterogeneous teaming problem as an MINLP optimization and approximate it as an MILP using sample average approximation. However, the number of variables and constraints in the linear program is a function of the sample number n_ξ. When the sample number dominates, the size of the linear program is roughly $\mathcal{O}(n_\xi)$, and the computation complexity is $\mathcal{O}(n_\xi^2)$.

We explore the sparsity of the problem and decouple it into a two-stage linear program using the L-shaped algorithm [99]. The algorithm returns the same optimal solution but reduces the computation cost from $\mathcal{O}(n_\xi^2)$ approximately to $\mathcal{O}(n_\xi)$ with a larger constant coefficient empirically.

The general structure of the algorithm is summarized in Algorithm 1. At the first stage, the algorithm solves a fixed-sized mixed-integer linear program (the size is not a function of the sample number n_ξ). At the second stage, the algorithm solves n_ξ small linear programs and then adds cuts to the first stage program according to the second-stage solutions. The two-stage process is iterated until convergence.

Algorithm 1 L-shaped Algorithm for the Model.

Input: Parameters of the optimization problem (2.3)-(2.21)
for $i \in M$ and $a \in A$ **do**
    $L_{ia} = 0$
while True **do**
    Solve the first stage problem and let $(y_{1i}^{[p]}, \cdots, y_{n_v i}^{[p]}, \lambda_{ia}^{[p]}, \theta_{ia}^{[p]}, x_{kij}^{[p]})$ be the solution.
    flag = True
    for $i \in M$ and $a \in A$ **do**
        // L_{ia} is abbreviated to L in the following lines.
        for $\xi = 1 : n_\xi$ **do**
            Solve the second stage problem (2.24) and let $\pi_{ia}^{(\xi)[p]}$ be the Lagrangian multiplier associated with the solution $w_{ia}^{(\xi)[p]}$.
        Calculate $D_{ia}^{[L]}$ and $d_{ia}^{[L]}$ according to (2.25).
        if $D_{ia}^{[L]}[y_{1i}^{[p]}, \cdots, y_{n_v i}^{[p]}, \lambda_{ia}^{[p]}]^\mathsf{T} + \theta_{ia}^{[p]} < d_{ia}^{[L]}$ **then**
            flag = False
            $L = L + 1$
            Add the cut $D_{ia}^{[L]}[y_{1i}, \cdots, y_{n_v i}, \lambda_{ia}]^\mathsf{T} + \theta_{ia} \geq d_{ia}^{[L]}$
    if flag **then**
        return the solution $x_{kij}^{[p]}$ // Optimality obtained.
    $p = p + 1$ // The problem will contain more cuts (2.23).

The first stage problem in the algorithm is

$$\min C_e \sum_{k \in V} \sum_{i,j \in N} b_{kij} \cdot x_{kij} + C_q \sum_{i \in U} q_i + C_h \sum_{i \in M} h_i \tag{2.14}$$

s.t. (2.3)-(2.12) and

$$h_i = \sum_{a \in A \text{ s.t. } \gamma_{ia} \neq 0} h_{ia}, \qquad \forall i \in M, \tag{2.15}$$

$$h_{ia} \geq \eta_\beta \left(-c_{ka} + \gamma_{ia}\right) \cdot r_{ki}, \quad \forall k \in V, a \in A, \forall i \in M, \tag{2.18}$$

$$h_{ia} = \lambda_{ia} + \frac{1}{n_\xi (1-\beta)} \theta_{ia}, \qquad a \in A, \forall i \in M, \tag{2.22}$$

$$D_{ia}^{[\ell]}[y_{1i}, \cdots, y_{n_v i}, \lambda_{ia}]^\mathsf{T} + \theta_{ia} \geq d_{ia}^{[\ell]},$$
$$\forall \ell = 1, \cdots, L, \quad a \in A, \forall i \in M. \tag{2.23}$$

If capability $a \in A$ is noncumulative, h_{ia} is calculated according to (2.18). If capability $a \in A$

is cumulative, the large linear program described in (2.21) is replaced with (2.22)-(2.23). θ_{ia} is a lower bound helper variable for $\sum_{\xi=1}^{n_\xi} w_{ia}^{(\xi)}$ in (2.21), and is tightened iteratively with the cuts in (2.23) obtained from the second stage.

In one iteration, as y_{ki} and λ_{ia} are determined in the first stage and fixed temporarily, the large linear program in (2.21) can be decoupled into n_ξ small linear programs with analytic solutions. We call it the second stage problem:

$$\begin{aligned} &\min\ w_{ia}^{(\xi)} \\ &\text{s.t. } w_{ia}^{(\xi)} \geq -\sum_{k\in V} c_{ka}^{(\xi)} \cdot y_{ki}^{[p]} + \gamma_{ia}^{(\xi)} - \lambda_{ia}^{[p]}, \\ &\quad w_{ia}^{(\xi)} \geq 0. \end{aligned} \tag{2.24}$$

Once the second stage problem is solved during each iteration, an additional optimality cut can be added in (2.23) according to the Lagrangian multipliers (simplex multipliers) $\pi_{ia}^{(\xi)[p]}$ associated with the second stage solutions. The parameters of the new optimality cut $D_{ia}^{[L]}$ and $d_{ia}^{[L]}$, are calculated as follows. Note that L is an integer label for the new cut.

$$\begin{aligned} D_{ia}^{[L]} &= \sum_{\xi=1}^{n_\xi} \pi_{ia}^{(\xi)[p]} \left[c_{1a},\ \cdots,\ c_{n_v a},\ 1\right]. \\ d_{ia}^{[L]} &= \sum_{\xi=1}^{n_\xi} \pi_{ia}^{(\xi)[p]} \cdot \gamma_{ia}^{(\xi)}. \end{aligned} \tag{2.25}$$

The Lagrangian multipliers $\pi_{ia}^{(\xi)[p]}$ are obtained by solving the second stage dual problem (2.26),

and the solutions are shown in (2.27).

$$\max \left(-\sum_{k \in V} c_{ka}^{(\xi)} \cdot y_{ki}^{[p]} + \gamma_{ia}^{(\xi)} - \lambda_{ia}^{[p]} \right) \cdot \pi_{ia}^{(\xi)} \tag{2.26}$$

$$\text{s.t. } 0 \leq \pi_{ia}^{(\xi)} \leq 1.$$

$$\pi_{ia}^{(\xi)[p]} = \begin{cases} 0, \ -\sum\limits_{k \in V} c_{ka}^{(\xi)} \cdot y_{ki}^{[p]} + \gamma_{ia}^{(\xi)} - \lambda_{ia}^{[p]} < 0 \\ 1, \ \text{otherwise} \end{cases} . \tag{2.27}$$

2.4 Decompose Agent Flows into Paths

Solving the teaming problem in Sec. 2.3 using the algorithm in Sec. 2.3.5 provides the optimal solution to $x_{kij}, \forall k \in V, \forall i, j \in N$, i.e., the flows of different agent species. This section discusses an optimal algorithm to extract a routing plan for each individual agent from the agent flows.

The flow decomposition problem contains two steps: rounding and splitting. First, we round the agent network flow obtained from the MIP risk minimization model in the previous section. Secondly, we split the network flow into a set of agent routes to obtain a plan for each individual agent.

An example is shown in Fig. 2.5. The flow network of species 1 is extracted and rounded off in Fig. 2.5b. Then, in Fig. 2.5c, three individual agent paths with flows of 1 on the path edges are selected such that the sum of the flows for the three paths equals the rounded integer network flow in Fig. 2.5b. Note that in order to maintain the task requirements, we have to round up a fractional flow rather than round it to the nearest integer.

In short, requirements for the rounding problem include:

- Round up the agent flow to integers.
- Maintain the flow constraint. (E.g., if two agents enter a task, they should also leave the task.)
- The rounding should result in minimum additional energy cost. (There can be multiple

solutions with only the first two constraints.)

Requirements for the splitting problem include:

- The integer agent flow should be split into multiple individual routes.
- The most costly routes due to the current split should be minimized. (There can be multiple solutions with only the first constraint.)

There is no trivial solution to the rounding and splitting problems. Additional information and examples are given in Sec. 2.7 (appendix). In the following two sections, we formalize two optimization problems to determine the optimal flow decomposition.

2.4.1 Minimum Energy Cost Rounding

Formally, a flow network is a graph $G(N = S \cup M \cup U, E)$ with a function $f(\cdot) : E \to \mathbb{R}$, where S, M, U, and E denote the set of start nodes, intermediate nodes, terminal nodes, and directed edges, respectively. For a node $m \in M$, suppose the incoming and outgoing edge sets are E_m^{in} and E_m^{out}, respectively. Then, the function $f(\cdot)$ should satisfy the flow constraint

$$\sum_{e \in E_m^{\text{in}}} f(e) = \sum_{e \in E_m^{\text{out}}} f(e), \quad \forall m \in M.$$

In this section, the flow function $f_k(i, j) = x_{kij}$ indicates the number of agent species $k \in V$ on edge $(i, j) \in E$.

See Fig. 2.5b as an example. Suppose the flow output from the MIP model in Sec. 2.3 for agent species $k \in V$ is x_{kij} on edge (i, j). Let the integer flow after the rounding process be x'_{kij}. Then,

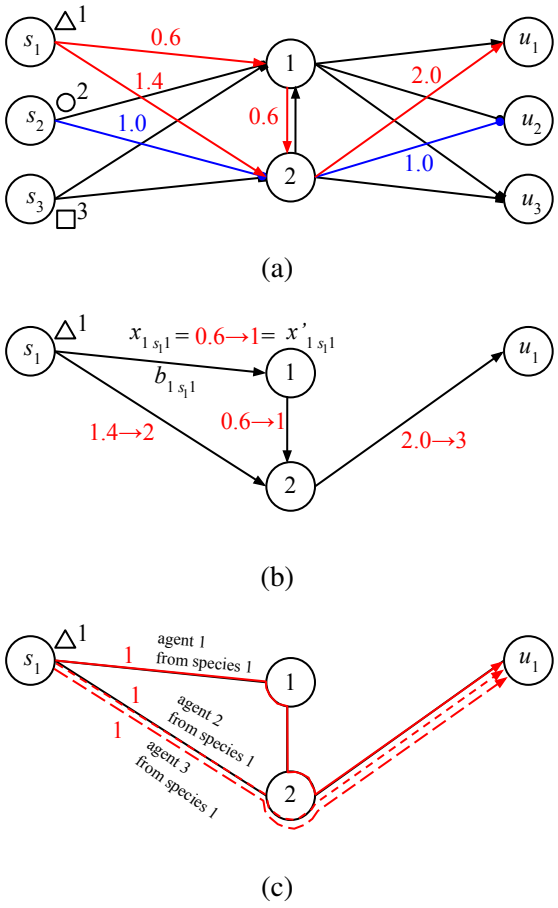


Figure 2.5: Decompose a flow into paths. The x's are the flow on an edge, with the red and blue numbers an example of the actual value. The colors distinguish the agent species. The b's are the energy cost of the edge. (a) Resulted agent flows. (b) The flow of agent species 1. (c) The split of agent species 1 using three individuals.

the linear program in below will return an integer flow network with minimum energy cost.

$$\min \sum_{i,j \in N} b_{kij} \cdot x'_{kij} \tag{2.28}$$

$$\text{s.t. } x'_{kij} \geq \lceil x_{kij} \rceil, \qquad \forall x_{kij} > 0,$$

$$x'_{kij} = 0, \qquad \forall x_{kij} = 0. \tag{2.29}$$

$$\sum_{i \in S \cup M} x_{kim} = \sum_{j \in U \cup M} x_{kmj}, \qquad \forall m \in M. \tag{2.30}$$

The objective function (2.28) penalizes the energy cost. (2.29) ensures the rounding is happen-

ing upward, where $\lceil x_{kij} \rceil$ denotes the smallest integer larger than x_{kij}. The network flow constraint (2.30) should be maintained during the optimization. S, U, and M are the set of start, terminal, and task nodes, respectively.

Note that there is no explicit integer constraint to ensure that x'_{kij} is an integer. However, if this linear program is solved using Simplex-based algorithms, the solutions to x'_{kij} are guaranteed to be integers. The proof is given in Sec. 2.8 (appendix). Because of this, the minimum energy rounding problem could be solved in polynomial time through a linear program (instead of an integer linear program).

2.4.2 Minimum Max-Energy Flow Split

After rounding, we obtain the integer flows x'_{kij} in Fig. 2.5b, the next step is to split this integer flow into individual agent paths in Fig. 2.5c. By summing the out-going flow at the start node, we are able to get the needed number of agents from species $k \in V$, denoted as n'_k. Then we compose n'_k graphs as in Fig. 2.6 for the n'_k agents.

We can formalize an integer linear program (ILP) as follows; find the routes for all agents, such that the maximum individual energy cost is minimized and the unit agent flow on the paths sums to the original network flow.

$$\min_{x^l_{kij}} \max_{l} \sum_{i,j \in N} b_{kij} \cdot x^l_{kij} \tag{2.31}$$

$$\text{s.t.} \sum_{l=1}^{n'_k} x^l_{kij} = x'_{kij}, \quad \forall i, j \in N, \tag{2.32}$$

$$\sum_{i \in N} x^l_{kim} = \sum_{j \in N} x^l_{kmj}, \quad \forall m \in M, \ \ \forall l = 1, \cdots, n'_k, \tag{2.33}$$

$$\sum_{i \in M} x^l_{ksi} = 1, \quad \forall l = 1, \cdots, n'_k, \tag{2.34}$$

$$x^l_{kij} \in \{0, 1\}, \quad \forall i, j \in N, \quad \forall l = 1, \cdots, n'_k.$$

In the ILP above, the objective (2.31) penalizes the maximum energy cost of an individual agent.

Equation (2.32) ensures that the resulting agent flows in sum up to the rounded flow. Equation (2.33) is the network constraint. Equation (2.34) requires that in each subgraph, there is only one agent.

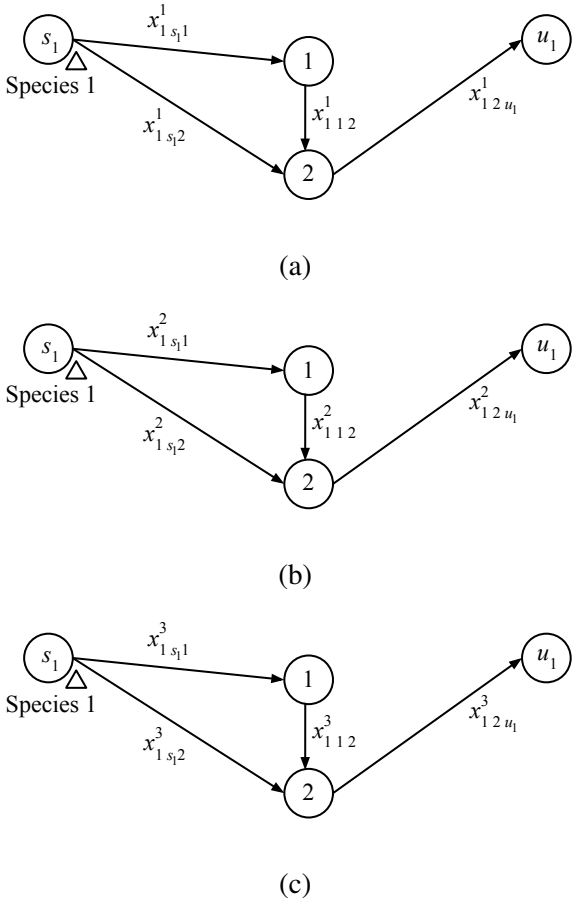


Figure 2.6: Graphical model for the minimum max-energy flow split problem for agent species 1. The x's are the flow on an edge. (a)(b)(c) are the flow networks for the three agents of species 1.

2.5 Experiments and Results

In this section, we first initialize randomized flow networks to test the decomposition of the agent flows, and then use two practical applications to evaluate the robustness, generalizability, and scalability of the proposed risk minimization model. The models and algorithms are implemented using the GUROBI solver. All computations were done on a laptop with Intel i7-7660U CPU

(2.50GHz).

2.5.1 Decompose Agent Flows into Paths

In this section, we use randomized test cases of different sizes to evaluate the computational cost of the flow decomposition process. The evaluation metric is the time to solve both the rounding and splitting problems to zero optimality gaps.

2.5.1.1 Setup and Test Cases

An agent flow network with random connections is initialized, and the flow and unit cost of each edge is sampled from uniform random distributions. The hyper-parameters are the maximum flow on an edge and the node number of the network. The optimization models in Sec. 2.4 are then applied to the initialized random flow network to solve the minimum energy cost rounding and minimum max-energy flow split problems.

2.5.1.2 Result and Discussion

Different sizes of flow decomposition problems are solved, and the computational costs of the two steps are listed in TABLE 2.3. The size of the rounding and split problems are proportional to edge number and edge number $\times$ sum integer flow, respectively. For the cases shown in TABLE 2.3, the rounding steps can be completed within several milliseconds. The rounding process scales well because the linear program can be solved in polynomial time. The split step involves solving an integer linear program, which does not scale well generally with the problem size. The largest test cases shown in the table involve 70 tasks and 241 agents on average, much larger than the typical teaming problem sizes that the risk minimization model will be applied to. Therefore, the flow decomposition part will not be the bottleneck of the overall teaming planner.

Though there is no explicit integer constraint in the rounding model in Sec. 2.4.1, we provided a proof in Sec. 2.8 (appendix) that showed that the solutions would be integers, and the results support it. As an example, the rounding solution for a case from the first row of the table is shown

Table 2.3: The computational cost of the flow decomposition.

Task*	Edge	Flow	Int flow	Round time	Split time
5	11±1	110.6±33.7	114±34	0.001±0.0005	0.09±0.04
10	28±3	21.3±2.8	30±2	0.001±0.0004	0.13±0.09
10	26±3	213.7±30.5	221±31	0.001±0.0002	1.23±0.89
35	107±8	135.7±13.2	172±13	0.001±0.0001	19.33±9.50
70	210±7	163.6±7.5	241±8	0.001±0.0009	197.0±32.9

* 'Task' is the number of tasks, which equals the number of nodes in the flow network. 'Flow' is the sum of agent flows from the start to the terminal node and 'int flow' is the value after rounding (i.e., the number of agents used). 'Round time' and 'split time' are the times used to solve the rounding and splitting problems, with units in seconds. For each row, we randomly initialize 10 similar sized test cases to evaluate the algorithm and show the mean and standard deviation.

in Fig. 2.7, where we can see that the agent flow on each edge is rounded up to an integer, while the overall network flow constraint is maintained. For instance, the 13.41 on edge (S, 1) is rounded to 15 instead of 14 in order to maintain the flow constraint.

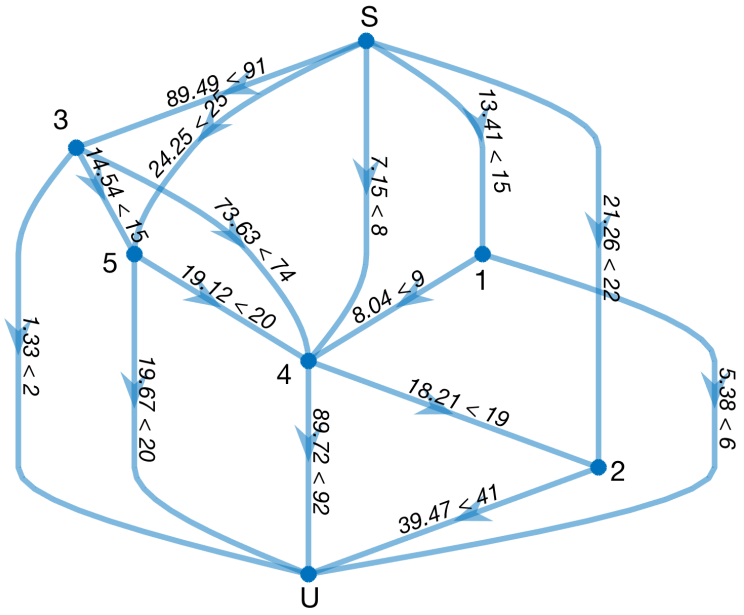


Figure 2.7: Flow rounding results (5 task nodes, 15 edges). There are two numbers on each edge, indicating the agent flows before and after the rounding process. This example corresponds to one instance in the first row of TABLE 2.3.

2.5.2 Capture the Flag

In this section, we apply the risk minimization model in Sec. 2.3 to a team of agents in a capture the flag game setting and compare the team performance against the baseline (STRATA [81]) in a simulation environment shown in Fig. 2.8. The goal of this simulation is to evaluate and demonstrate the performance of the task assignment component of our framework. The number of wins is used as the metric for task performance.

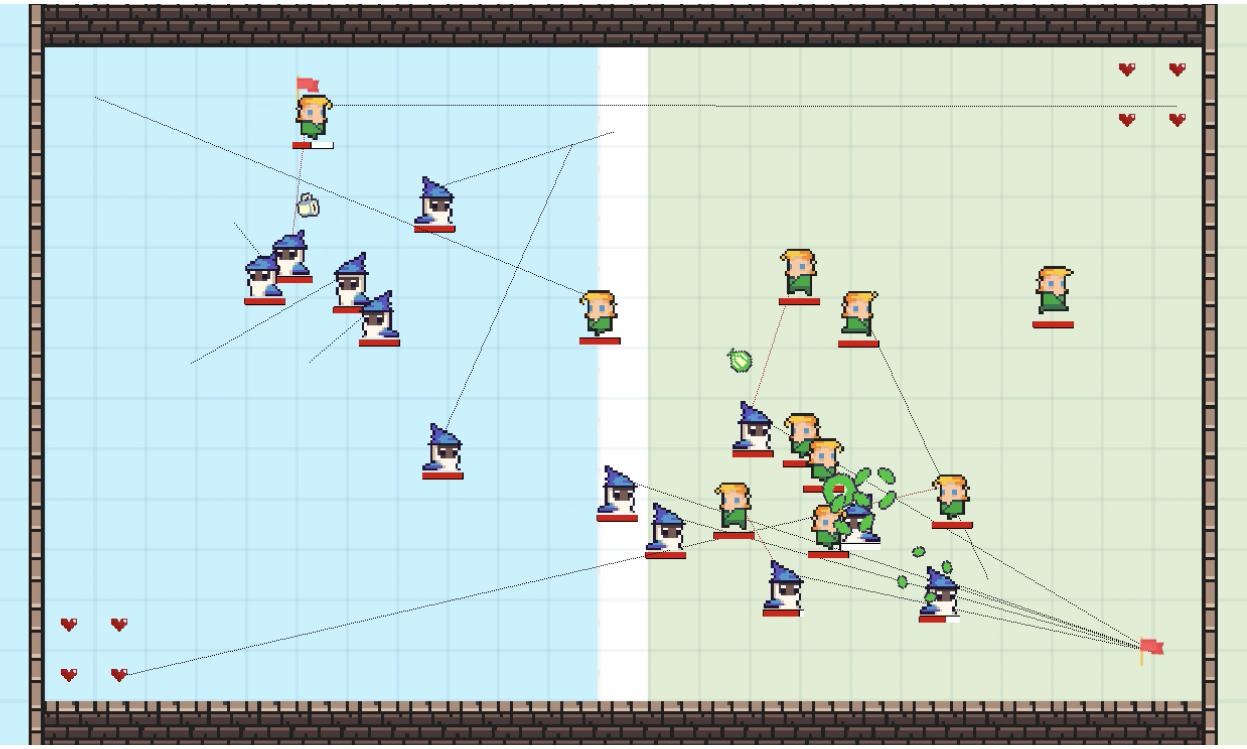


Figure 2.8: Capture the flag setup. Each small square tile is 1×1 unit length. A line points to the current target of the corresponding agent.

2.5.2.1 Game Setup, Baseline, and Metrics

The blue and green teams contain 12 heterogeneous agents initialized at random locations within their own sides. The overall goal is to win the game by capturing the flag from the other team. The 12 agents are from 4 species (3 individuals for each species). These species have different speed, viewing distance, health, and ammunition capabilities (TABLE 2.4). The specific values

in the table are set according to the baseline paper [81]. Among these capabilities, the first two are modeled as noncumulative, while the last two are cumulative.

Each agent is designed to play one role out of the 3 tasks listed in TABLE 2.5 and will disappear when its health reaches zero. The adversarial elements of the game are not explicitly modeled, and the focus is to assign tasks such that the requirements are fulfilled while overall time and energy costs are considered in the trade-off. Therefore, the agents follow predefined behaviors once their tasks are determined.

STRATA [81] is chosen as the baseline for this work because its model makes use of similar concepts like agents, species, capability vectors, and task requirements specified by capabilities. Since there is no scheduling involved in this game (the tasks are all 'life-long' tasks), CTAS reduces to an instantaneous task assignment, tackling the same type of problem as STRATA. Therefore, this game will provide a comparison between STRATA and the task assignment component of CTAS. In addition to STRATA, we also compare our model to a random task assignment mechanism where an agent is randomly assigned a task.

All games are played with two teams using different assignment models. The metric used in this simulation is the number of wins in 500 games, which reveals the relative task assignment performance. A win is either a team taking back the other side's flag or defeating all enemies through a fight mode. A draw happens when all of the agents from both sides decide to defend and no longer attack, or if no team wins the game within 120 seconds.

Both STRATA and our model need task requirements specified as capability distributions. For the three tasks, the agents are programmed to switch to a healing mode automatically when their health is lower than a threshold and return to attack/defend after they heal. Namely, the healing role is a temporary state that happens when we think an agent is too 'unhealthy' to continue. In the game, CTAS and the baseline assign either the attack or defend role to the agents. The threshold of entering the healing state is set to 33% of an agent's full health. We have done a parametric study, and results show that the threshold does not affect the number of wins and the task assignment patterns that will be discussed. The required capabilities of the attack and defend tasks are listed

in TABLE 2.6. The task requirements in this game are deterministic. However, since the agent capabilities in Table 2.4 are stochastic, the planning problem is still stochastic.

The specific numbers are manually decided to reflect the following requirements: the attack task needs agents with higher speed and health to capture the flag, while the defend task needs agents with higher viewing distance and ammunition to detect and defeat the opposing team. In TABLE 2.6, 1131 is 65% of the expected total initial health of the 12 agents, and 231 is 55% of the expected total ammunition.

Table 2.4: Capability distributions of the four agent species in the capture-the-flag game.

species	μ_1*	μ_2	μ_3	μ_4	σ_1^2	σ_2^2	σ_3^2	σ_4^2
1	1.5	2	90	40	0.35	0.1	10	3
2	1.5	4	60	40	0.35	0.1	10	3
3	3	2	80	30	0.35	0.1	10	3
4	3	4	350	30	0.35	0.1	10	3

* μ_i and σ_i^2 ($i = 1$: speed, 2: view, 3: health, 4: ammunition) are the means and variances of the capability distributions (Gaussian in the experiment).

Table 2.5: General task descriptions of the capture-the-flag game.

Task	Description
Attack	Try to capture the flag of the other side.
Defend	Apply a defense mode to defeat opposing team members.
Heal	Use hearts to apply healing powers.

Table 2.6: Task requirements specified as capability distributions.

	Speed	View	Health	Ammunition
Attack	≥ 2		≥ 1131 (65%)	
Defend		≥ 1		≥ 231 (55%)

2.5.2.2 Results

Based on all the settings described in the above section, we developed a simulation environment in C++. A screenshot is shown in Fig. 2.8. The three task assignment models, random, baseline, and CTAS, are linked with the simulation environment. During the simulation, the mean time for the baseline and CTAS to optimize and output an assignment are 0.43 and 0.04 seconds, respectively.

The relative performances of the models are shown in Fig. 2.9. As can be seen from Fig. 2.9, CTAS results in a higher win rate as compared to the baseline or random selection methods. The average number of agents assigned to attack, grouped by species, is shown in Fig. 2.10. Both the baseline and CTAS prefer using species 4 for the attacking task with the difference being that the baseline still uses species 1 and 2 for attack occasionally.

Though the baseline claims to be able to consider noncumulative capabilities through thresholding, it lacks an explicit mechanism to prevent incompetent agents from joining a task. For instance, though species 1 and 2 are not competent for the attack task due to their low speed, they are still allowed to conduct the task and contribute to other required capabilities such as health. However, this is not a good choice as there exist other agents that satisfy both the speed and health requirements of the attack task.

Another possible factor contributing to CTAS's improved performance is that the CTAS algorithm directly minimizes the CVaR metric, which ensures enough task-required capabilities, whereas the baseline focuses on matching the expected requirements and penalizing variance of the assigned capability distributions.

In conclusion, through the comparison, our framework demonstrates superior task assignment performance against the baseline algorithm, and the task assignment patterns in Fig. 2.10 support this performance result.

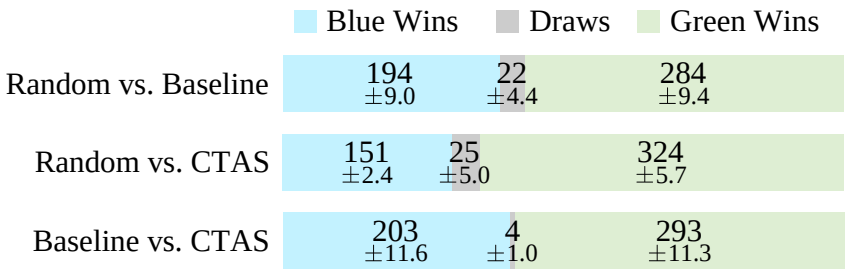


Figure 2.9: Comparison between the task assignment performance using random assignment, baseline, and CTAS. Each row shows the mean and standard deviation from ten 500 capture the flag games. The length of the bars are proportional to the mean.

2.5.3 Robotic Services during a Pandemic

In this section, we demonstrate the generality and the task assignment and scheduling components of our framework by considering a more complex scenario inspired by the COVID-19 pandemic. We use our previous work in [44] as a baseline. The energy cost and probability of success will be used to evaluate the optimality of the generated plans. The optimality gap given limited time for the optimization will be used to evaluate the scalability of the framework.

2.5.3.1 Experiment Setup and Model Description

Consider pandemic robotic services in a city environment consisting of multiple delivery, disinfection, test, and treatment sub-tasks that require one or multiple agents to complete. The task-required capabilities and agent capabilities are modeled as random distributions. We choose to include 8 types of tasks and 7 agent species in our investigation, some of the agents needing a human operator.

Applying the CTAS model to describe the problem of the pandemic robotic services, we first define 9 capability types according to the chosen tasks and agents in TABLE 2.7.

The 7 agent species and their capabilities are defined in TABLE 2.8, according to real-world references [100]. The columns are different capabilities. The agent capabilities are Gaussian distributed random variables in this example. The values in the table are expectations of the random distributions, and the standard deviations are 10% of the expectations.

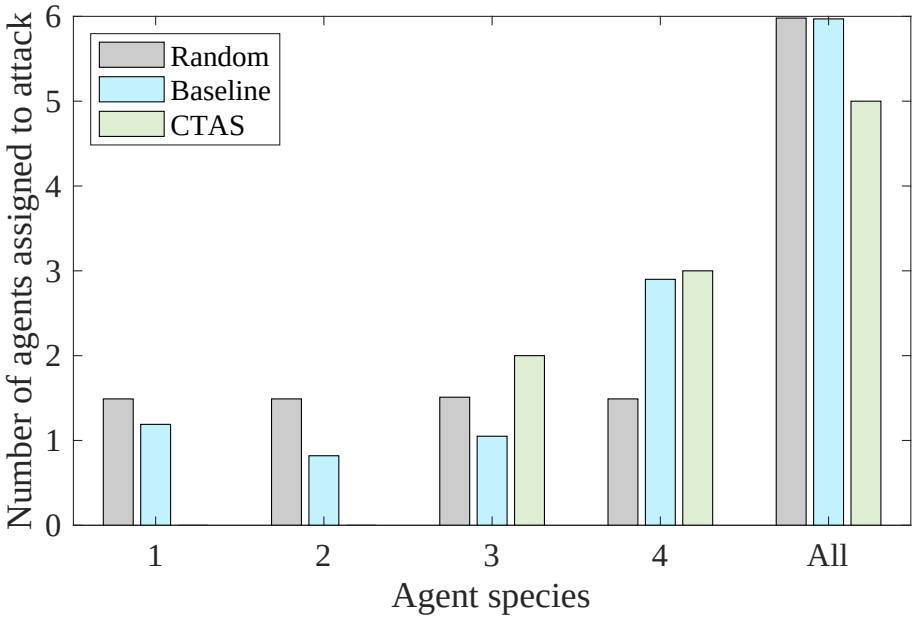


Figure 2.10: The average number of agents assigned to attack. CTAS is more likely to use species 4 and does not use species 1 and 2. The baseline shows a similar trend of preferring species 4 over the other species. This preference is logical since species 4 contains attributes of high speed and health, which are more suitable for the attack task than other species' attribute distributions (particularly species 1 and 2).

The 8 task types are described in TABLE 2.10. Their required capabilities are listed in TABLE 2.9. The columns are different capabilities. A value in the table is the expected requirement for that specific capability, and the standard deviation is 10% of the expectation. γ is a scaling coefficient. E.g., the γ in (row 1, column 3) means task type 1 requires the team's capability $\alpha_3 \geq \gamma$. When γ is set larger, more agents get involved in the optimization and the problem space is larger. We do not scale noncumulative capabilities. For one task, the requirements on different capability types are imposed with 'and' logic. These tasks are distributed in a city. An example of 16 tasks (two tasks from each type) is shown in Fig. 2.17a. The agents start from the base, which, in practice, can be a hospital.

We use the M3500 dataset [101] as the city's road map in this illustration. We assume that we have a viral exposure level (a real number) at some locations and can infer a cost map of the virality level using Gaussian process [44, 102] based on the assumption that the neighborhood of a high-cost location is also high-cost. For example, in the city map in Fig. 2.17a, we randomly choose 6

Table 2.7: Definitions of the capability types in the robotic service example.

Cap	Definition
1	Fly (noncumulative)
2	Equipped with a freezer
3	Deliver materials
4	Conduct perception
5	Remove and collect harmful materials
6	Conduct viral test
7	Physically interact and conduct treatment
8	Spray disinfectant
9	Place signals and barricades to conduct quarantine enforcement

contaminated and 6 proven-safe regions as samples and learn a viral exposure map. The blue and red regions have low and high cost, respectively, while the white regions have less information, high ambivalence, and are assigned a medium cost. Based on this, we compute a viral-exposure-based travel cost (A* path [103]) between the tasks and regard it as the energy cost for the edges in the graph in Fig. 2.4.

We choose the agents and tasks from their types in TABLEs 2.8 and 2.9 and compose 32 mission cases where the agent numbers, task numbers, and the coefficient γ are chosen from {21, 70, 140}, {16, 24, 32, 40}, and {1, 3, 5, 10}, respectively. For each mission case, we generate 6 instances with randomly selected tasks locations, which result in different graphical models in Fig. 2.4. For the test cases, task i and $i + 8$ have the same requirements, but are at different locations.

Based on these cases, we evaluate the mission performance and computational cost of the three models in TABLE 2.11:

- CTAS: The risk minimization model
- CTAS-D: The risk part, h_i, is removed from the objective function (2.14).
- CTAS-O: Solve the same problem as CTAS-D without the flow decomposition. The size of the math problem is larger and harder to solve. This is our baseline from [44] to compare.

Table 2.8: Stochastic agent capabilities in the robotic service example.

Agent species	1	2	3	4	5	6	7	8	9
1: Quadcopter	1		1	1				1	
2: Vehicle			1						2
3: Vehicle (freezer)		1	1						
4: Vehicle (contaminants)					1			1	
5: Guidance robot				1					5
6: Test robot						1			
7: Treatment robot						1	1		

Table 2.9: Stochastic task requirements in the robotic service example.

Task type	1	2	3	4	5	6	7	8	9
1: Goods delivery			γ						
2: Goods delivery (fly)	1		γ						
3: Medical kits delivery		γ	γ						
4: Contaminants removal					γ			γ	
5: Open area disinfection				γ	γ			2γ	
6: Quarantine enforcement			γ	γ	γ			γ	10γ
7: Viral tests						γ			
8: Remote treatment						γ	γ		

2.5.3.2 Computational Cost and Discussion

A 120-second time limitation is added to the solvers of the models for all test cases. For each mission case, we run the 6 test instances and if the solver can find a feasible solution within the time limit, we consider it successful. The success rate of each model is shown in Fig. 2.11. We show the detailed average performances for the mission sizes with success rates larger than 50% in Fig. 2.12-2.15. The optimality gaps after the rounding process in Sec. 2.4.1 are given in Fig. 2.12a. The blue cells mean the solver's success rate for the specific mission size is less than 50% for the time limit picked. According to the two figures, CTAS-D and CTAS can solve much larger teaming problems than CTAS-O. The largest problem that CTAS-D solves in this example involves 140 agents and 40 tasks.

The increases in the optimality gaps due to the rounding process are shown in Fig. 2.12b. For

Table 2.10: Task descriptions in the robotic service example.

Task type	Description
1: Goods delivery	Deliver goods.
2: Goods delivery (fly)	Deliver goods using quadcopters.
3: Medical kits delivery	The team should be equipped with freezers.
4: Contaminants removal	Remove contaminated materials and then disinfect the location.
5: Open area disinfection	Perceive a contaminated area to conduct contaminants removal and disinfection.
6: Quarantine enforcement	Perceive, carry materials to, disinfect a contaminated area, and then place signals and barricades to enforce a quarantine.
7: Viral tests	Conduct viral tests.
8: Remote treatment	Conduct viral tests and treatment.

Table 2.11: Three teaming models. 'Agent var' stands for 'agent variables'.

Model	Agent var	Objective	Variable number
CTAS-O [44]	Binary	Energy + Time	Large
CTAS-D	Real	Energy + Time	Relatively small
CTAS	Real	Energy + Time + Risk	Relatively small

most of the cases, the rounding process introduces an increase in the optimality gap within 1%. The leftmost cases have larger increases because the agent numbers are small, and rounding has a larger overall influence. Since CTAS-O involves no rounding process, the increased gap is 0 for all test cases.

In these test cases, the agent capabilities are Gaussian distributed. Therefore, given the estimation of the means and variances, we can calculate the probability of success and its mean as follows

$$P(\text{task } i \text{ succeeds}) = \prod_{\text{capability } a} P(a \text{ satisfied}) \quad \forall i \in M$$

$$\text{mean } P(\text{success}) = \left(\prod_{i \in M} P(\text{task } i \text{ succeeds}) \right)^{1/n_m}$$

The mean probabilities of success for the three models are shown in Fig. 2.13. For all cases, CTAS-O and CTAS-D models roughly result in a probability of 0.25. This is because each task requires two capabilities on average. When the risk is not considered, these two models tend only to match the expected requirement, and the probability of matching a single capability is 0.5. Clearly, the risk minimization model increases the probability of success for the tasks.

In Fig. 2.14, we compare the result of CTAS to its deterministic version CTAS-D. With the chosen β[3], C_e, and C_h[4], for most of the cases, a $\sim$20% increase in energy cost introduces a $\sim$35% increase in the mean probability of success.

The trade-off between energy cost and robustness in the objective function and can be tuned smoothly. The trade-off gained by changing the C_e, C_h, and β is shown in Fig. 2.16, using the smallest test case as an example. The penalty coefficient has a major impact on the trade-off between energy and the probability of success, while β has a local impact on the trade-off. According to our investigation, choosing $\beta = 0.8 - 0.97$ would mostly cover the Pareto set. This shows the advantage of using the CVaR as a risk metric: better Pareto optima are gained than optimizing the expectation ($\beta \to 0$).

For the sample average approximation of the CVaR, we use 500 samples to approximate the Gaussian distributions. Given the solution, we compare the nonlinear objectives to their sample approximations and find that the relative approximation gaps are smaller than 1% for most of the cases, as shown in Fig. 2.15. This shows that the approximation quality is good using 500 samples.

2.5.3.3 Mission Performance and Discussion

This section uses a test case with 16 tasks, 21 agents, and $\gamma = 1$ as an example to compare the solutions generated by the three models. In TABLE 2.12, we list the performance metrics of 5 tasks whose success rate is increased by the CTAS model. All three models are solved to optimal within the time limit. The models presented in this chapter end up with much fewer variables for the same problem. By comparing CTAS with CTAS-D, we see that the risk minimization model reduces the

[3] β: hyperparameter in the CVaR function $\eta_\beta(\cdot)$ in Fig. 2.3 and objective function (2.16) and (2.17).
[4] C_e and C_h: the penalty coefficients on energy cost and task completion in equation (2.14).

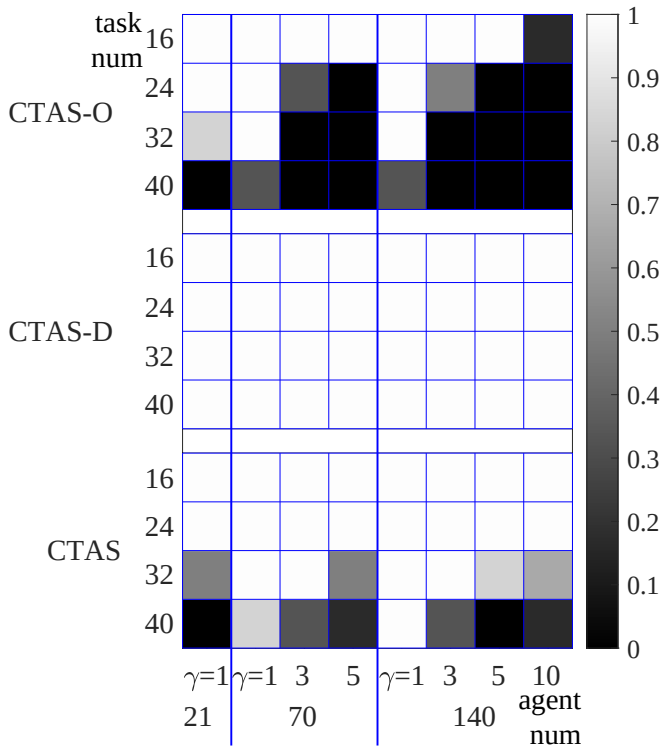


Figure 2.11: The success rate of the models for each mission test case (6 instances are tested for each case). The grayscale colors correspond to the value specified in the color bars on the right.

CVaR and increases the probability of success for 5 tasks out of 16, with a small increase (5.3%) in the overall energy cost.

The teams for the last 8 tasks are shown in TABLE 2.13. In the table, v_i stands for one agent from species i. As expected, the CTAS model puts more agents in the team to reduce the CVaR. This task assignment results from simultaneously considering the energy cost. As an example, CTAS puts more agents in the team of tasks 9-12, but not tasks 1-4, even if they are of the same task types. Because energy and time costs are jointly considered, and tasks 1-4 are far from the depot, adding more agents to ensure redundancy and robustness could potentially result in much higher costs. The risk minimization model also generates the routes and a consistent schedule. For example, one agent from species 1, 4, and 5 visit task 13 at the same time as a team. The routes of species 1, 4, and 5 are shown in Fig. 2.17. These routes minimize overall travel distances and avoid the high-cost red regions to lower energy costs.

In summary, according to the teams in TABLE 2.13 and routes in Fig. 2.17, the CTAS frame-

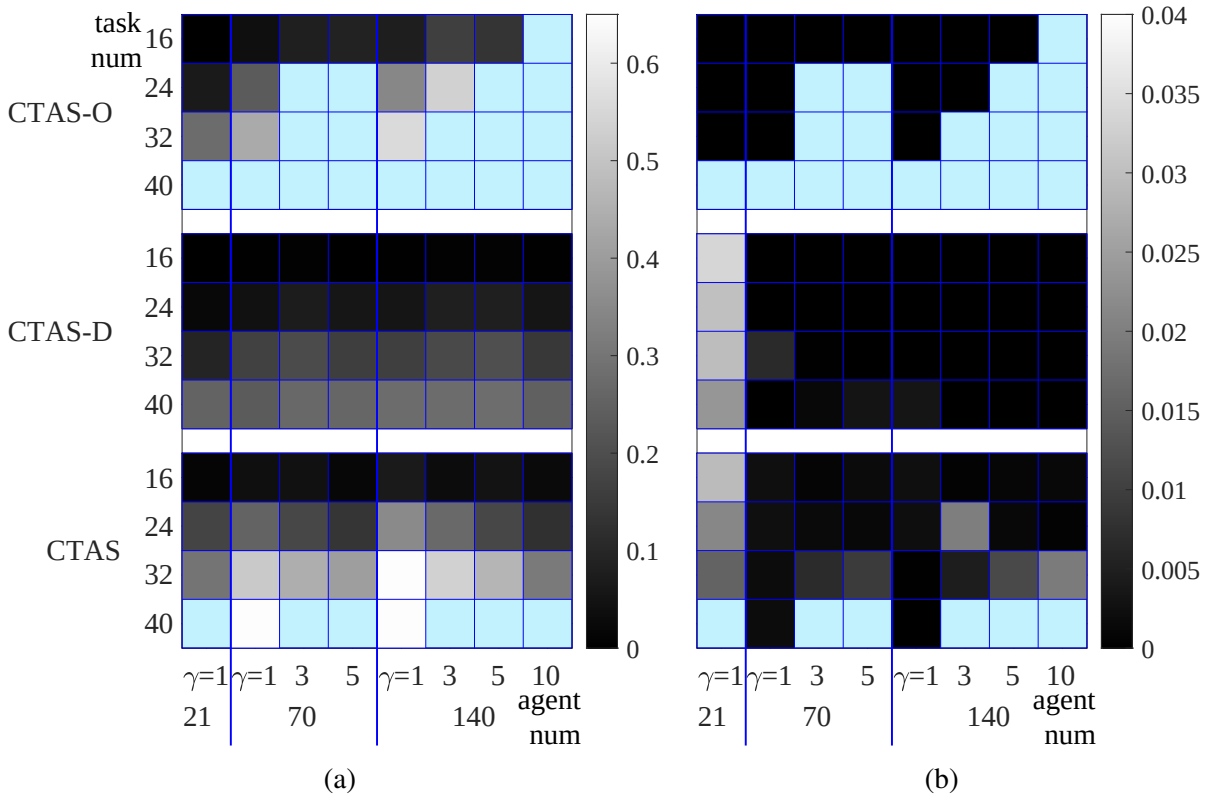


Figure 2.12: The optimality gap of the three models applied to the 32 test cases, defined as (objective value - lower bound) / lower bound. A smaller optimality gap indicates a better solution. The grayscale colors correspond to the value specified in the color bars on the right. The **blue cells** mean the solver's success rate for the specific mission case is $< 50\%$ for the time limit picked (120 seconds). (a) The optimality gap before the flow rounding process. The two white cells in the CTAS group are outliers whose values are around 1.0. (b) The increased optimality gap due to the rounding process.

work generates a consistent schedule for coordination, outputs routes that minimize energy costs, and assigns tasks to agents such that redundancy is preserved at low costs to ensure a higher probability of task completion. The computational evaluation in Sec. 2.5.3.2 (particularly, Fig. 2.12) shows that the frameworks CTAS and CTAS-D scale to 140 agents and 40 tasks with low optimality gaps. The scalability in agent number is better than the task number. Furthermore, CTAS-D still shows no optimality gap degeneration dealing with the largest test case we tested.

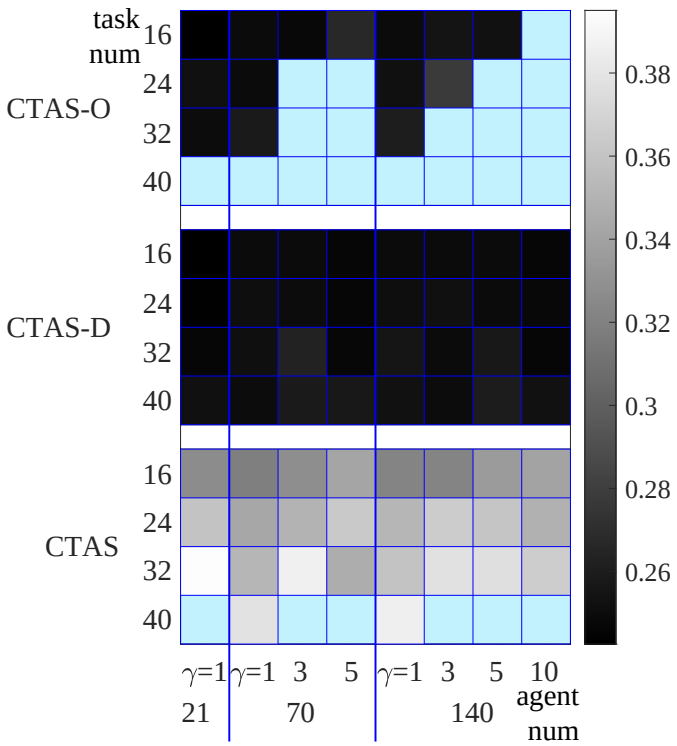


Figure 2.13: The mean probability of success for the tasks using the three models.

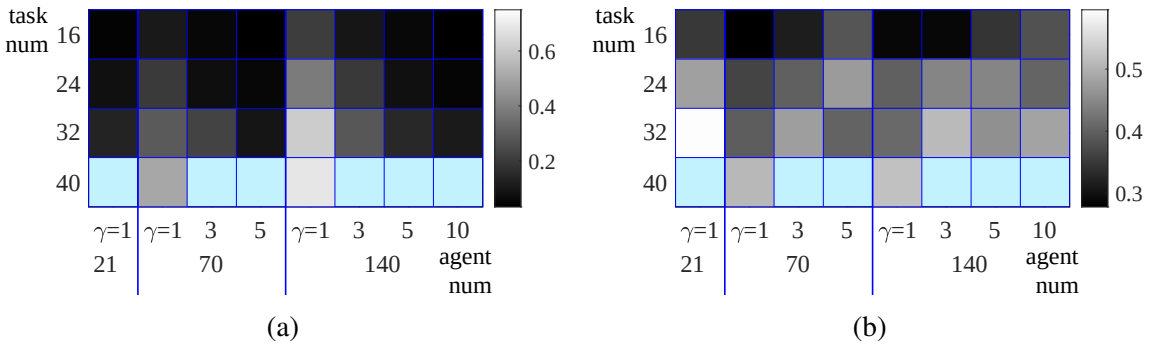


Figure 2.14: Comparing the results of CTAS to CTAS-D. The values (colors) in the girds are (a) increased energy (relative) and (b) increased mean probability (relative) by adding the risk as an objective. I.e., the values are (CTAS − CTAS-D) / CTAS-D.

2.6 Conclusions and Future Work

The framework proposed in this chapter addresses a complex task allocation problem in the category of CD-[ST-MR-TA][5]. We propose a mixed-integer programming model that simultane-

[5]The category of task allocation problem: complex task dependencies (CD), single-task robot (ST), multi-robot tasks (MR), and time-extended assignment (TA).

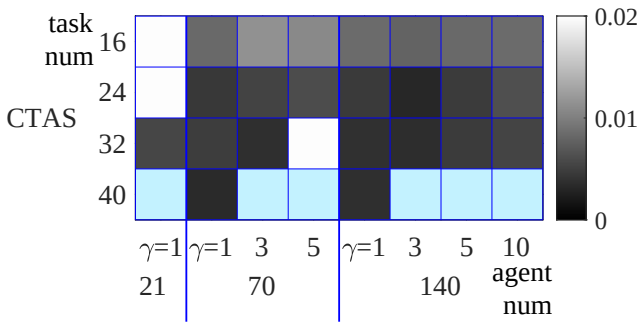


Figure 2.15: The relative approximation gap of the nonlinear CVaR through sample average approximation and linear programming.

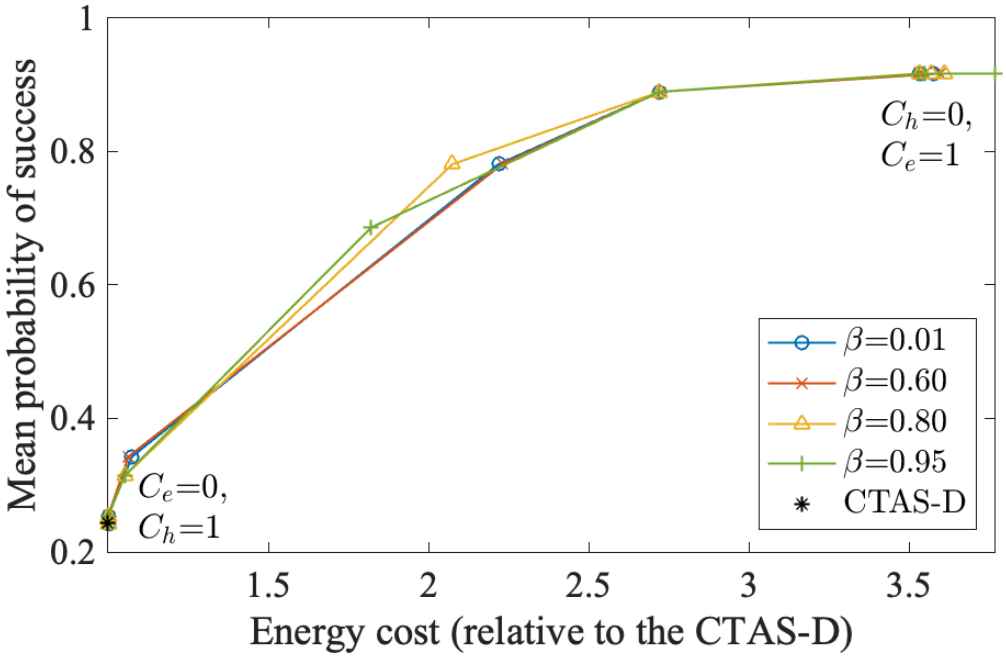


Figure 2.16: Trade-off between energy cost and risk of task's non-completion for a case with {16 tasks, 21 agents, and $\gamma = 1$}. The upper-left corner is the direction of the Pareto set.

ously optimizes the task decomposition, assignment, and scheduling. The uncertainty within the team's capability is considered through risk minimization, and a robust metric, conditional value at risk (CVaR), is minimized to ensure the robustness to task completion (enough capability are assigned). The framework contributes to a domain-independent representation for complex tasks and heterogeneous agent capabilities that can generalize to multi-agent applications where the major goals are satisfying task-required capabilities. A two-step solution method is described, and the

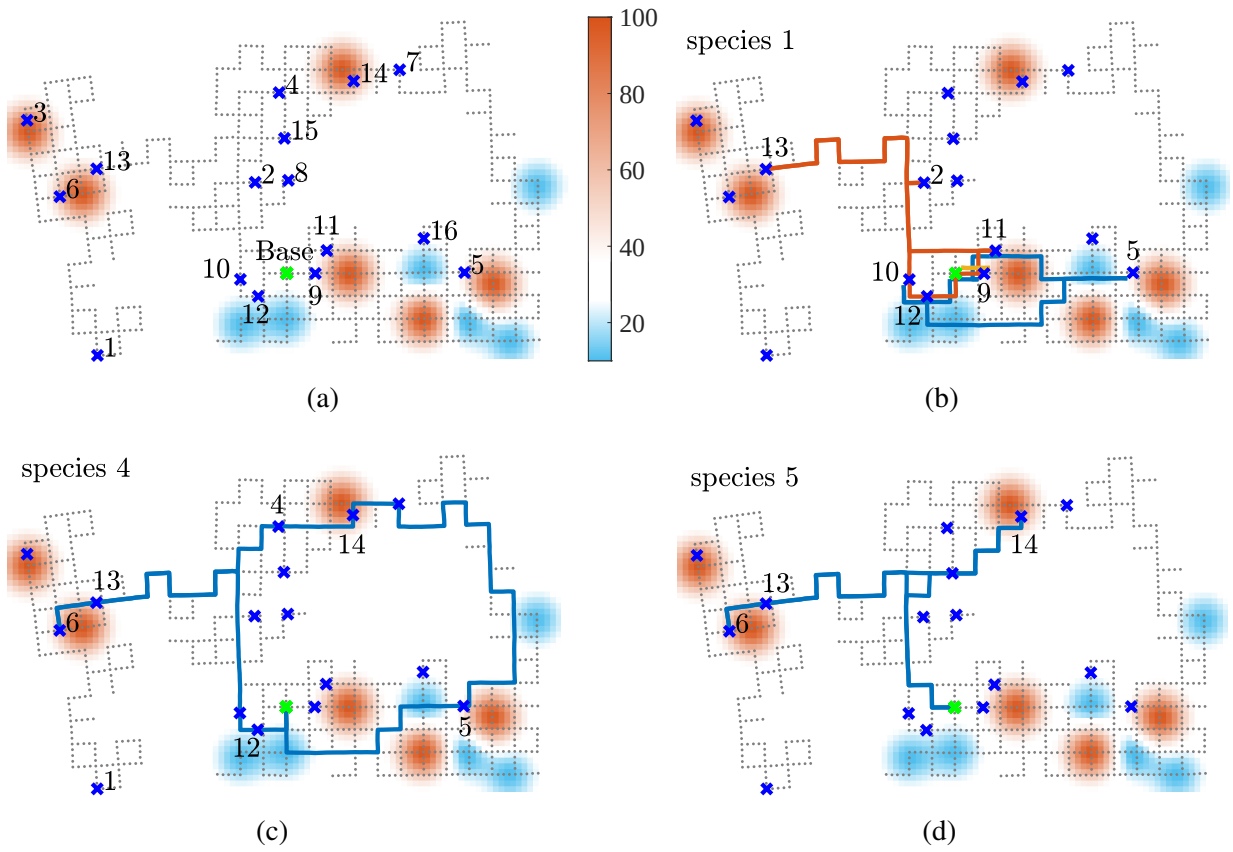


Figure 2.17: Routes for the robotic services. (a) Task distribution: the 16 tasks are distributed in a city [101] where the unit travel cost depends on viral exposure. Blue and red stand for low and high energy costs, respectively. (b)-(d) The planned routes from the CTAS model for species 1, 4, and 5. Different colored lines represent distinct agent individuals from the same species.

whole framework is evaluated in two different practical test cases. Results show that the framework scales up to 140 agents and 40 tasks for the cases tested and solves the problems with low optimality gaps. Given the selected hyper-parameters, the resulting assignments and schedules provide a reasonable trade-off between energy, time, and the probability of success. The task assignment performance (apart from the scheduling) is also demonstrated through a comparison with the STRATA framework in the capture the flag case.

Future work will consider an extension of this work to a distributed framework, which could further improve the scalability of the system and the robustness to communication constraints and the loss of agents. For modeling choices, we will consider imposing a necessary and sufficient energy constraint in the CTAS model in Sec. 2.3 while still preserving the scalability of the cur-

Table 2.12: A comparison of the results of the three teaming models.

Item	CTAS-O	CTAS-D	CTAS
Variables	6194	4276	4384
Task CVaR	5.41 $\times 10^3$	5.41 $\times 10^3$	-1.03 $\times 10^4$
Energy cost	2.06 $\times 10^5$	2.06 $\times 10^5$	2.17 $\times 10^5$
$P(\text{task } 9)$	0.5	0.5	1
$P(\text{task } 10)$	0.25	0.25	0.5
$P(\text{task } 11)$	0.25	0.25	1
$P(\text{task } 12)$	0.25	0.25	0.5
$P(\text{task } 13)$	0.125	0.125	0.25

Table 2.13: Team configurations for the robotic services given by the three models.

Task	CTAS-O	CTAS-D	CTAS
9	v_3	v_3	$v_1 \times 3, v_2 \times 3, v_3 \times 3$
10	v_1	v_1	$v_1 \times 2$
11	v_3	v_3	$v_1 \times 2, v_2 \times 3, v_3 \times 3$
12	v_4	v_4	$v_1 \times 2, v_4$
13	v_1, v_4	v_1, v_4	v_1, v_4, v_5
14	$v_2 \times 3, v_4, v_5$	$v_2 \times 3, v_4, v_5$	$v_2 \times 3, v_4, v_5$
15	v_7	v_7	v_7
16	v_7	v_7	v_7

rent framework. In addition, this chapter assumes the task requirements and agent capabilities are given. In practice, such model parameters are often unknown or contain significant uncertainty. Therefore, a probabilistic learning method that automatically estimates the parameters in the representation of task requirements and agent capabilities from current and previous task executions is an interesting future work. Such a learning method would enable the possibility of closing the loop of the task assignment and scheduling, and iteratively improving the performance. We will address this research gap in Chapter 4.

2.7 Appendix: Non-trivialness of the Flow Decomposition

A rounding or splitting without solving an optimization can break flow constraints or result in suboptimal solutions.

The rounding process has to maintain the flow constraints: for all nodes, the incoming flow must equal the outgoing flow. A naive rounding might break the flow constraint. Take the node 3 in Fig. 2.18a as an example: naively rounding up the flows on an edge will result in 2 incoming agents, but 3 outgoing agents.

The rounding selection process is not unique, even when meeting the flow constraints. For instance, Fig. 2.18b-c are both valid ways to round the flow in Fig. 2.18a. However, the energy cost of the flow in Fig. 2.18c is higher. Therefore, the optimal rounding process will need to maintain the flow constraint and introduce the lowest additional energy cost.

After rounding the flow to integers, there are multiple choices to assign individual agent paths. We call this 'splitting the flow into paths'. For example, the flow in Fig. 2.18b can be split into the 3 agent routes either in Fig. 2.19a or Fig. 2.19b. Though the total energy costs are equivalent, the energy costs of the three paths in the two choices are {20, 20, 20} and {20, 24, 16}, respectively. The choice affects the behavior of an individual agent. Here, we prefer the former split, because the energy costs of individual agents are more evenly distributed and the maximum energy cost of an individual agent is smaller. We define the optimal flow split as the set of paths that minimizes the maximum individual energy cost.

2.8 Appendix: Proof of the Integer-valued Solutions

Definition 1: The incidence matrix for a directed graph is a matrix of n_{node} rows and n_{edge} columns: for each row, if an edge enters the node, the matrix value is 1; if an edge leaves the node, the value is -1; otherwise 0.

Theorem 1: The incidence matrix of any directed graph is a totally unimodular (TU) matrix [104, 105].

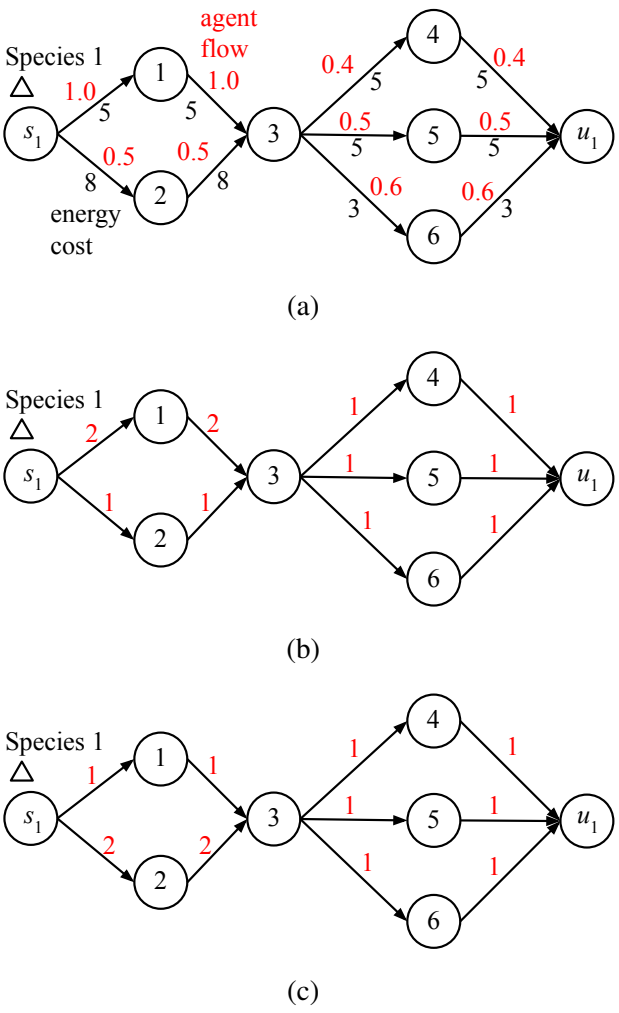


Figure 2.18: Round the flow of agent species 1. The red number is the agent flow on an edge. The black number below an edge is the energy cost of the edge. (a) The original flow with real numbers. (b) Rounding choice 1. (c) Rounding choice 2.

Theorem 2: If A_1 is a TU matrix and E is an identity matrix, then $A = [A_1|E]$ will be a TU matrix [104].

Theorem 3: A is a TU matrix iff A^{T} is a TU matrix (by definition of a TU matrix [104]).

Theorem 4: If A is a TU matrix and $\mathbf{b}$ is integer-valued, then each vertex of the polytope defined by $A\mathbf{x} \leq \mathbf{b}$ is integer-valued [104].

Proof. **The solutions of the rounding problem from Simplex-based algorithms will be integer-valued.**

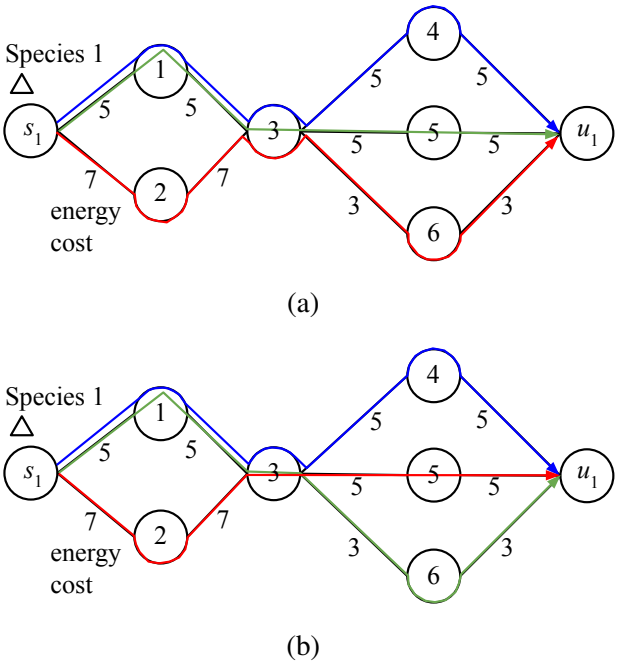


Figure 2.19: Split the agent flow in Fig. 2.18b into 3 paths (colored). The black number below an edge are the energy cost of the edge. There are two choices with different individual costs. (a) Choice 1. (b) Choice 2.

Now we write the constraints of the rounding problem, i.e., (2.29)-(2.30), in the form of $A\mathbf{x} \leq \mathbf{b}$. See Fig. 2.20, suppose the flow constraint (2.30) is converted into $A_1\mathbf{x} \leq \mathbf{b}_1$, and the bound constraint (2.29) is converted into $A_2\mathbf{x}_1 \leq \mathbf{b}_2$.

According to **Definition 1**, A_1 is an incidence matrix of a directed graph. $\mathbf{b}_1$ is a zero vector. According to the format of constraints (2.29), A_2 is an identity matrix, and $\mathbf{b}_2$ is an integer-valued vector. Therefore, $\mathbf{b}$ is integer-valued.

According to **Theorem 1**, A_1 is a TU matrix.

According to **Theorem 3**, A_1^T is a TU matrix.

Because A_2^T is an identity matrix, according to **Theorem 2**, $[A_1^\mathsf{T}|A_2^\mathsf{T}]$ is a TU matrix.

According to Fig. 2.20 and **Theorem 3**, $A = [A_1^\mathsf{T}|A_2^\mathsf{T}]^\mathsf{T}$ is a TU matrix.

Because matrix A is a TU matrix and $\mathbf{b}$ is integer-valued, according to **Theorem 4**, all vertices of the polytope defined by $A\mathbf{x} \leq \mathbf{b}$ are integer-valued.

If Simplex-based algorithms are applied to the optimization problem, the algorithms search

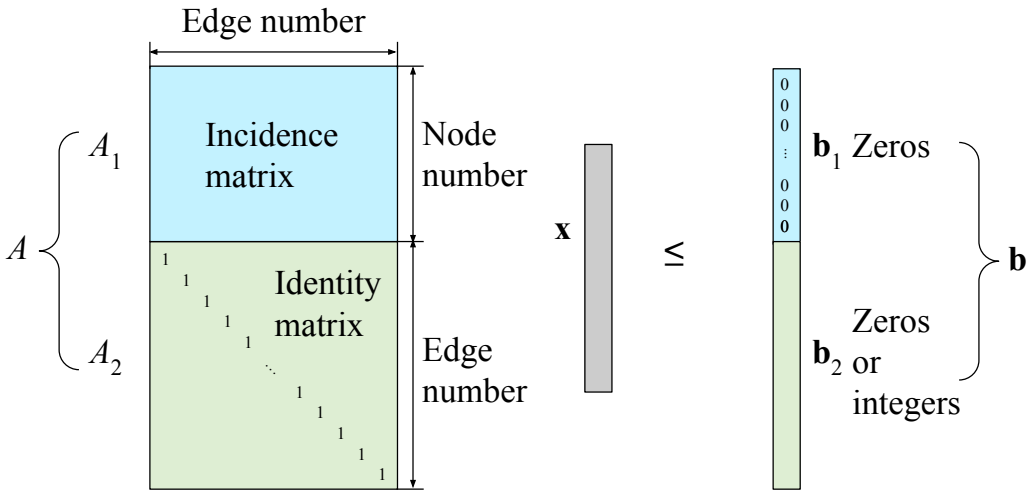


Figure 2.20: Totally unimodularity of the rounding problem.

through vertices of the polytope to find the optimal point. Therefore, the optimal point must be a vertex of the polytope. Since we have proved that all vertices are integral, the solution to $\mathbf{x}$ will be integral. □

CHAPTER 3

Human-robot Matching and Routing for Multi-robot Tour Guiding under Time Uncertainty

In the previous chapter, we developed a framework to solve a complex task allocation problem under uncertainties in the task requirements and agent capabilities. In this chapter, we show that similar techniques can be used to deal with a multi-robot task allocation where a part of the decision making is affected by human factors. In addition, time uncertainty, a different type of uncertainty as compared to chapter 2, is considered. The task assignment, scheduling, and routing for the agents are still coupled. Therefore, this chapter supplements Chapter 2 to comprehensively address contribution 1 in Fig. 1.1.

This chapter presents a framework for multi-robot tour guidance in a partially known environment with uncertainty, such as a museum. In the proposed centralized multi-robot planner, a simultaneous matching and routing problem (SMRP) is formulated to match the humans with robot guides according to their selected places of interest (POIs) and generate the routes and schedules for the robots according to uncertain spatial and time estimation. A large neighborhood search algorithm is developed to efficiently find sub-optimal low-cost solutions for the SMRP. The scalability and optimality of the multi-robot planner are evaluated computationally under different numbers of humans, robots, and POIs. The largest case tested involves 50 robots, 250 humans, and 50 POIs. Then, a photo-realistic multi-robot simulation platform was developed based on Habitat-AI to verify the tour guiding performance in an uncertain indoor environment. Results demonstrate that the proposed centralized tour planner is scalable, makes a smooth trade-off in the plans under

different environmental constraints, and can lead to robust performance with inaccurate uncertainty estimations (within a certain margin).

The remainder of this chapter is organized as follows: Sec. 3.1 motivates the multi-robot tour guiding task and presents it as a simultaneous matching and routing problem. Sec. 3.2 briefly introduces work related to socially assistive robots and the SMRP problem. Sec. 3.3 describes the math formulation and algorithms of the SMRP planner for multi-robot tour guides. Sec. 3.4 introduces the local navigation planner and the simulation environment. Note that sections 3.3-3.4 also describe how uncertainties are considered in the planner and simulated in the simulation. Sec. 3.5 provides computational and simulation results, followed by discussion. Sec. 3.6 concludes the findings of the chapter with potential extensions.

This chapter is adapted from our paper [95].

Supplementary videos and open-source code for this project can be found at:
https://brg.engin.umich.edu/publications/asme2021tourguide.

3.1 Introduction

Robot tour guides have been applied in different environments [106–111] because they save human labor, do not require time-consuming training, and have the possibility to ease discomfort during social interactions [112]. Previous works have been focusing on the routing of places of interest (a traveling salesman style problem) [113], localization and navigation within the environment [110, 111], and social interactions with humans through gesture, face, and language processing [114]. While they cover diverse topics, these works mainly focus on single robot guiding for a small predetermined team. In practice, it is unlikely that a single robot will be used to guide all of the human visitors; hence there is a need to use multiple robots to handle a larger number of humans. In such a situation, instead of considering each robot separately (as in previous works), a crucial question is how to coordinate the robots as a multi-robot system, such that the overall efficiency/performance of the system is maximized.

A natural first step is to consider loose coordination: once the humans are split into teams and assigned to each robot, the single-robot tour guide systems in previous works can be applied to the low-level navigation and interaction tasks. However, the problems of choosing the best robot guides for a human and finding the optimal tour plan for a robot depend on each other. The two problems are tightly coupled, leading to challenges to the optimization of both the tour plan and the human-robot matching.

Note that in this formulation, we consider homogeneous robots, and therefore, the local navigation and social interaction behaviors of the robots are homogeneous, do not affect, and therefore, are not considered during the multi-robot coordination at this time. Future work may consider navigation and complex social interactions within the matching and touring problem.

This work focuses on a multi-robot planning system that simultaneously optimizes the human-robot team and a high-level robot tour plan specified as a route of places of interest (POIs). The system provides an optimal matching between humans and robots and the corresponding tour plans. The optimization planner proposed in this chapter can be paired with a local navigation/interaction planner to form a complete framework that can be used to handle practical multi-robot guidance applications. Additionally, the touring system can be applied to different environments, such as a museum docent, a park tour, and a city tour.

As the multi-robot planning is performed premission, the actual touring times may vary due to dynamic movements of the teams, changes in the environment, and uncertain visiting times at a POI. Such variations will be considered as uncertainty in the traveling and visiting time in the optimization to generate time-robust touring plans.

A computational investigation is conducted with the proposed multi-robot planning system. Then, the system performance in an uncertain environment is evaluated through a photo-realistic simulation. A system diagram of the simulation framework is shown in Fig. 3.1. A centralized multi-robot tour planner first generates the teams and tour plans for all robots. Then, individual robots use a local navigation planner to follow the planned routes and lead the humans within the simulation environment.

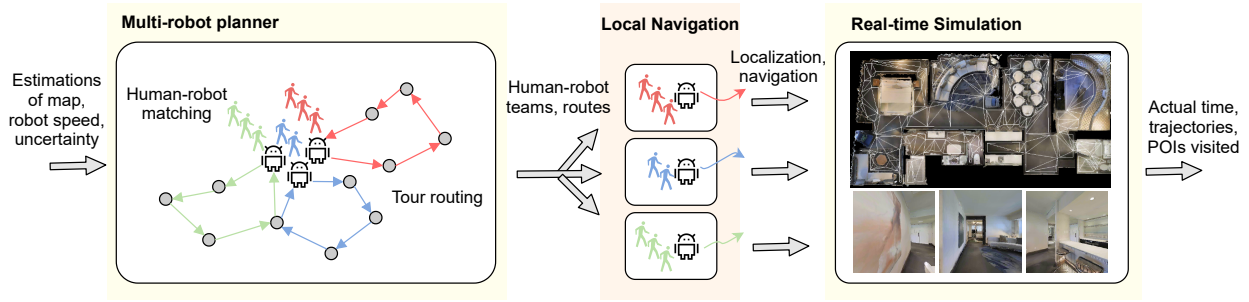


Figure 3.1: A framework for applying our multi-robot planner for tour guiding within a simulation. The centralized multi-robot tour planner is discussed in Sec. 3.3. The local navigation and simulation are described in Sec. 3.4.

This work has the following contributions.

1) The algorithmic modeling of a **simultaneous** human-robot **matching** and tour **routing problem (SMRP)** for multi-robot tour guidance under execution time uncertainty.

2) A comprehensive computational evaluation of the scalability and solution quality of the proposed algorithms.

3) The simulation verification of the proposed multi-robot system through a concrete tour guiding case study in a photo-realistic indoor environment.

3.2 Related Work

In this work, the multi-robot tour guiding problem contains not only tour planning but also the human-robot team generation. Therefore, the optimization combines two classic problem types from the operations research: bipartite matching and vehicle routing. The matching problem refers to splitting the humans into teams such that the matching between humans and robots minimizes the number of dropped human requests. It is a bipartite matching problem as there are two separate sets of elements (robots and humans) in the problem. The definition of bipartite matching in the tour planning is illustrated in Fig. 3.2. The routing problem focuses on optimally arranging the robot tours (routes), such that the dropped requests, as well as the traveling distance, are minimized

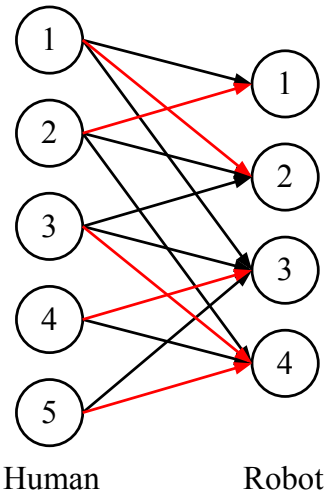


Figure 3.2: A bipartite matching example. A bipartite graph is a graph where there are two sets of nodes and no edge connections within each set. A matching is a selected set of edges that satisfy some desired properties. In this work, a matching should ensure that every human node is matched to one robot node (the red edges form a matching). The bipartite matching problem in the tour planning tries to find the best matching between humans and robots, such that the overall satisfied human requests are maximized.

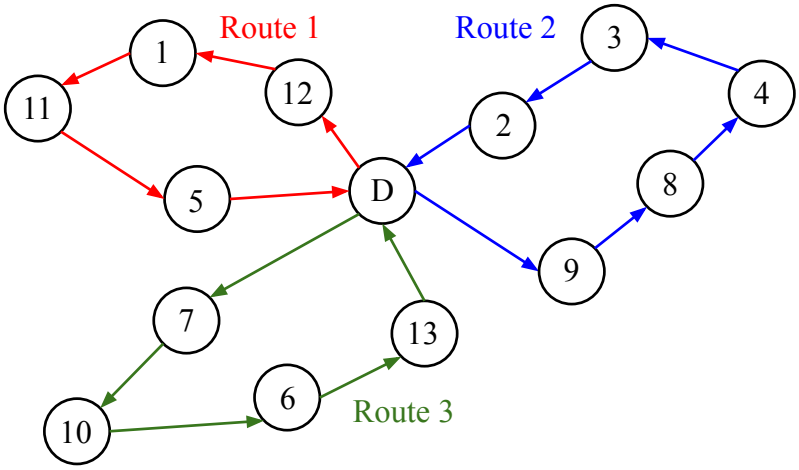


Figure 3.3: A vehicle routing problem (VRP) example with three vehicles. The VRP tries to find the minimum-cost vehicle paths that satisfy some user-defined constraints.

(see Fig. 3.3).

Optimal bipartite matching can be found through linear programming or maximum flow optimization [115]. Bipartite matching has been applied to multiple real-world problems including image feature matching [116], object detection [117], budget allocation [118], and task assignment

[119].

The vehicle routing problem (VRP) considers the minimum traversing distance of all the places of interest using multiple vehicles (robots). It generalizes the traveling salesman problem and is NP-hard to solve optimally. The stochastic vehicle routing problem (SVRP) is a variation of VRP where some of the parameters in the optimization are random distributions. Previous works in the field of SVRP have investigated optimization under uncertain requests number at each POI [45, 120–122], uncertain time costs for traveling edges or service at a POI [27, 123–125], and uncertain energy costs [44, 126, 127]. This work considers the uncertainty in time costs.

In this work, the optimal robot guide for a human depends on the robots' routes, while the route of a robot depends on the humans in its team. Therefore, the matching and routing problems are coupled to form a larger problem that must be optimized simultaneously. An SMRP problem contains multiple vehicle routing problems (formally, a VRP can be reduced to an SMRP); and since a VRP is NP-Hard [128], an SMRP is NP-hard. A ride-sharing system can be regarded as an SMRP where the riders should be matched with drivers while the routes are determined simultaneously. However, many systems in previous work decouple the problems by generating the vehicle routes first and then matching humans with the closest routes [129, 130]. Another way to formulate the ride-sharing problem is as a VRP with pickup and delivery [131]. Optimally solving the simultaneous matching and routing optimization under uncertainty can be challenging as this is a discrete non-convex problem that is NP-hard. Unlike previous work, we directly address this simultaneous optimization problem, model it as a mixed-integer optimization, and provide methods that efficiently find sub-optimal low-cost solutions in Sec. 3.3.

3.3 Human-robot Matching and Tour Routing

Consider the following practical situation: several human visitors arrive and each selects several places of interest. The guiding system wants to use a number of robots to guide these humans around the environment while satisfying as many POIs as possible. It is assumed that there are

fewer robots than humans, and therefore, the people need to be split into teams. Since there is a time limit for the tours, not all human requests can be satisfied. There is uncertainty in the traveling and visiting time during the tour. A multi-robot planning system needs to find an optimal way to form the human-robot teams and the corresponding tours (defined as routes and schedules for the teams) to minimize the dropped requests under the aforementioned constraints and time uncertainties.

In this section, we first formally define the simultaneous matching and routing problem for the above practical multi-robot tour guidance. Then, we encode the problem as a mixed-integer program and provide a heuristic algorithm to solve the problem.

3.3.1 Problem Description and Graphical Model

Suppose there is a set of guidance robots $V = \{1, \cdots, n_V\}$, a set of humans $L = \{1, \cdots, n_L\}$, and a set of POIs to visit $M = \{1, \cdots, n_M\}$. Each human can choose a subset of POIs that they want to visit. In this problem, a planner needs to determine which robot a human should follow and what routes (a sequence of POIs) a robot should take, such that the number of satisfied human requests (POIs) is maximized within certain touring time limits.

Other user-defined practical constraints include time window constraints (certain POIs are available only during fixed time windows), sequence dependencies (the visit of certain POIs has prerequisites that other POIs be visited first), and human dependencies (some people may prefer to be assigned to the same robots, such as families and friends).

Suppose the start and terminal locations of the robots are nodes (places) $n_M + 1$ and $n_M + 2$. Namely, we have two additional nodes $s = n_M + 1$ and $u = n_M + 2$ in the graph. Then, let $N = \{1, \cdots, n_M, s, u\}$ be the whole set of places. Note that, M is a subset of N, and M does not include the start and terminal. We first define a directed graph $G = (N, E)$, with N and E the sets of nodes and edges, respectively (see Figure 3.4). For each node pair $\{i, j\}$ with $i, j \in N$, we add two directional edges (i, j) and (j, i) between them. Therefore, we have the edge set denoted as $E = \{(i, j)\},\ \forall i, j \in N$. Note that in our model, there is no assumption that the POIs are on a flat

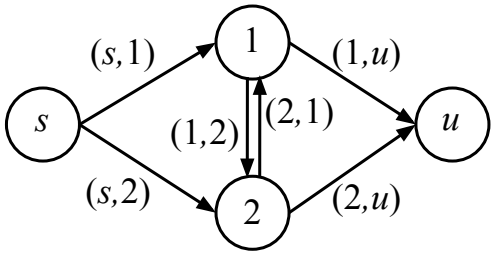


Figure 3.4: A graphical model example with two POIs. s and u are the start and terminal, respectively.

plane. If the tour happens in a multi-level building, no further modeling is needed as long as the edge costs are correctly adjusted according to the levels.

3.3.2 A Mixed-integer Bilinear Program

Here we discuss a mixed-integer bilinear program that mathematically models the SMRP we discussed above. We first provide the notations that will be used in Table 3.1. Note that the decision variables are x_{kij}, y_{ki}, t_{ki}, and z_{lk}. Other symbols in Table 3.1 are predefined hyper-parameters or parameters that describe the environment and human-related information. A graphical illustration of the key notations is shown in Figure 3.5.

Objective Function: In the objective function (3.1), we want to minimize the weighted combination of dropped human requests and time usage, with the weights C_a and C_t, respectively. The right part is the sum of the time that each human-robot team reaches the terminal node. The left part is the total number of dropped POIs for all humans. As an example, if human l follows robot k, z_{lk} equals 1. If this human l wants to see POI i, a_{li} equals 1. In addition, if robot k does not pick POI i in its route, y_{ki} equals 0. Finally, if $a_{li} = 1$, $z_{lk} = 1$, and $y_{ki} = 0$ happen simultaneously, the left part of the objective function will be incremented by 1, penalizing the drop of a human request. In this example, the weights are chosen such that $C_a \gg C_t$ to ensure that the main focus is on satisfying human requests.

$$\min \; C_a \cdot \sum_{l \in L} \sum_{k \in V} z_{lk} \left(\sum_{i \in M} a_{li} \cdot (1 - y_{ki}) \right) + C_t \sum_{k \in V} t_{ku}. \tag{3.1}$$

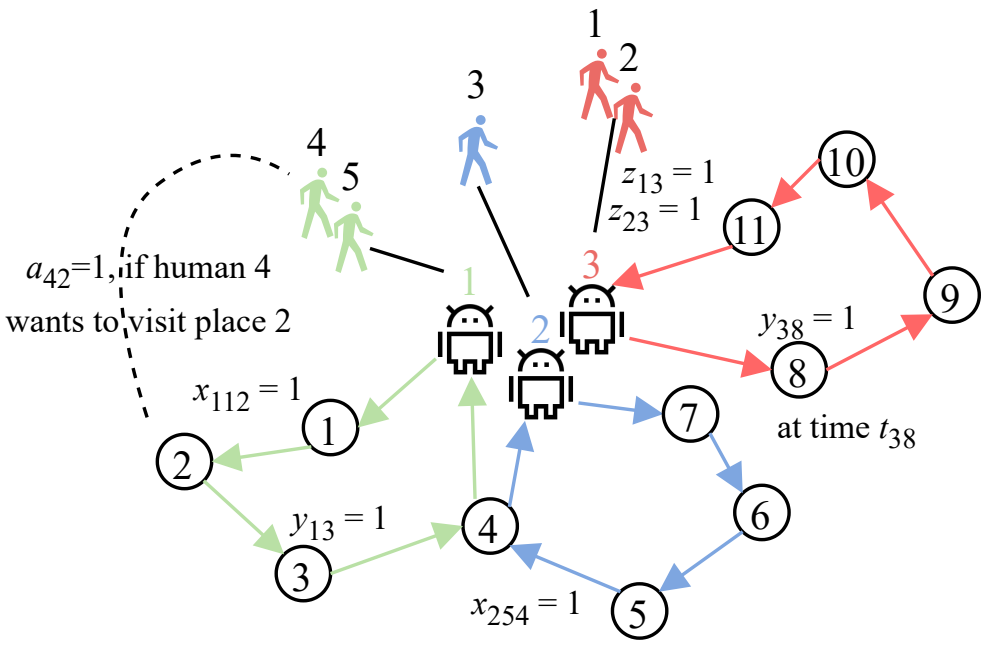


Figure 3.5: Key notations in a simultaneous matching and routing problem.

Variable Bounds: The feasible regions and bounds of four sets of decision variables are defined as in (3.2).

$$x_{kij},\ y_{ki},\ z_{lk} \in \{0, 1\}, \quad t_{ki} \geq 0,$$
$$\forall k \in V,\ \forall i, j \in N,\ \forall l \in L. \tag{3.2}$$

Network Flow Constraints: Equation (3.3) is a network flow constraint that ensures the incoming robot number equals the outgoing robot number. Constraint (3.4) ensures that there is only one path in the network flow of robot k. Constraints (3.5)-(3.6) show the relationship between variable x and y. The number of robots at a node equals the number of incoming robots through the corresponding edges. Moreover, the number of robots at any node should be no larger than the number departing

Table 3.1: Definition of the notations in a simultaneous matching and routing problem.

	Meaning
x_{kij}	= 1, if robot $k \in V$ travels edge $i, j \in N$, otherwise 0.
y_{ki}	= 1, if robot $k \in V$ visits node $i \in M$, otherwise 0.
t_{ki}	The time robot $k \in V$ visits node $i \in N$.
z_{lk}	= 1, if human $l \in L$ is assigned to follow robot $k \in V$, otherwise 0.
a_{li}	= 1, if human $l \in L$ wants to visit node $i \in M$, otherwise 0.
V	The set of robots.
L	The set of humans.
M	The set of POIs.
N	The set of POIs plus the start and terminal.
M_T	The set of POIs with additional time window constraints.
S	The set of sequence dependencies.
L_{pair}	The set of human pairs who want to be in the same team.
T_L	A large constant time.
T_{ki}	The time for robot k and its team to visit POI i.
T_{kij}	The time for robot k and its team to travel edge (i, j).
T_k^{lim}	A preset time limit for the tour guided by robot k.
$T_i^{\min}$	The lower bound for a preset time window for visiting POI i.
$T_i^{\max}$	The upper bound for a preset time window for visiting POI i.
$Z_k^{\max}$	The maximum number of humans guided by robot k.
$A^{\max}$	The maximum number of POIs that can be dropped for one human.

from the start node.

$$\sum_{i \in N} x_{kim} = \sum_{j \in N} x_{kmj}, \quad \forall m \in M, \quad \forall k \in V. \tag{3.3}$$

$$\sum_{i \in N} x_{ksi} \leq 1, \qquad \forall k \in V. \tag{3.4}$$

$$y_{kj} \leq \sum_{i \in N} x_{ksi}, \qquad \forall j \in M, \quad \forall k \in V. \tag{3.5}$$

$$y_{kj} = \sum_{i \in N} x_{kij}, \qquad \forall j \in M, \quad \forall k \in V. \tag{3.6}$$

Time Constraints: Inequalities (3.7)-(3.8) are cumulative time constraints: if robot k travels edge (i, j), then $x_{kij} = 1$, and the difference between the arrival time at node j and node i equals the total time for visiting node i and traveling edge (i, j) (i.e., $T_{ki} + T_{kij}$). Constraint (3.9) sets a time limit T_k^{lim} for the whole tour. Constraint (3.10) sets a specific time window $[T_i^{\min}, T_i^{\max}]$ for

visiting a node. This is important for some of the POIs that have a specific opening time (e.g., a film or a show). We use the set $M_T \subset M$ to denote these specific POIs.

$$t_{ki} - t_{kj} + T_{kij} + T_{ki} \leq T_L(1 - x_{kij}),\ \forall i, j \in N, \forall k \in V. \tag{3.7}$$

$$t_{ki} - t_{kj} + T_{kij} + T_{ki} \geq -T_L(1 - x_{kij}),\ \forall i, j \in N, \forall k \in V. \tag{3.8}$$

$$t_{ku} \leq T_k^{\lim}, \qquad \forall k \in V. \tag{3.9}$$

$$T_i^{\min} \leq t_{ki} \leq T_i^{\max}, \qquad \forall k \in V, \forall i \in M_T. \tag{3.10}$$

Sequence Dependency Constraints: There can be sequence dependencies between the POIs. For example, if a person does not visit POI i, they would not be able to gain the knowledge needed to understand and appreciate the work at POI j. Let S denote the set of the sequence dependencies. As an example, S can be denoted as $\{(1,2),(2,3),(6,8)\}$, which means that POI 2 depends on 1, and POI 3 depends on 1 and 2, while POI 8 has POI 6 as a prerequisite. Mathematically, the sequence dependency is encoded as in (3.11). The first inequality ensures that node i has to be visited if a route visits node j. The second inequality related to time ensures that node i is visited before j.

$$y_{ki} \geq y_{kj}, \quad t_{ki} \leq t_{kj} + T_L(1 - y_{kj}),\ \forall k \in V,\ \forall (i,j) \in S. \tag{3.11}$$

Team Size Limit Constraints: All humans must be matched to a robot guide, introducing the constraint (3.12). We also impose a limit to the maximum number of humans that follow a robot to avoid imbalanced teams as in (3.13). This limit is denoted as $Z_k^{\max}$. Due to this constraint, a human might not be matched to their optimal robot guide for balancing the assignment and minimizing overall dropped POIs. However, we add constraint (3.14) to limit the maximum number of dropped

requests, $A^{\max}$, to avoid extreme sacrifices of specific humans' needs.

$$\sum_{k \in V} z_{lk} = 1, \qquad \forall l \in L. \tag{3.12}$$

$$\sum_{l \in L} z_{lk} \leq Z_k^{\max}, \qquad \forall k \in V. \tag{3.13}$$

$$\sum_{k \in V} z_{lk} (\sum_{i \in M} a_{li} \cdot (1 - y_{ki})) \leq A^{\max}, \quad \forall l \in L. \tag{3.14}$$

Human Dependency Constraints: There can be human dependencies where some people want to be guided by the same robot guide (e.g., families and friends). Such a dependency is encoded in (3.15), which forces two people to be in the same team. L_{pair} is the set of human pairs that have such dependencies. An example can be $L_{\text{pair}} = \{(1,2),(2,3),(5,6)\}$.

$$z_{l_1 k} = z_{l_2 k}, \quad \forall k \in V, \ \ \forall (l_1, l_2) \in L_{\text{pair}}. \tag{3.15}$$

3.3.3 Uncertain Travel and Visiting Time

In the above formulation, we do not consider time uncertainty. When there is uncertainty in the traveling time T_{kij} and visiting time T_{ki}, we still need to make sure that the tour ends on time, i.e., constraint (3.9) still holds. In such situations, T_{kij}, T_{ki}, and t_{ku} can be modeled as random distributions. And we penalize the scenarios where the finishing time t_{ku} is close to the time limit $T_k^{\lim}$ by adding an expected penalty, $\phi_(t_{ku})$,

$$\begin{aligned}
\phi_(t_{ku}) =& \mathrm{E}\left([t_{ku} - \alpha_k]^+\right) = \lim_{n_\xi \to \infty} \frac{1}{n_\xi} \sum_{\xi=1}^{n_\xi} \left[t_{ku}^\xi - \alpha_k\right]^+, \\
\alpha_k =& T_k^{\lim} - \Delta T_k^{\lim}, \\
[x]^+ =& \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}.
\end{aligned}$$

α_k is a time threshold when the penalty starts being applied and $\Delta T_k^{\lim}$ is a time margin which is usually chosen as a small number (e.g., 5 minutes). t_{ku}^{ξ} is a sample of the distribution t_{ku} with $\xi = 1, \cdots, n_{\xi}$. Such a penalty model has shown effectiveness in previous work in VRP with stochastic time cost [27]. In practice, we only calculate an approximation of the $\phi(t_{ku})$ with a bounded number of samples as follows

$$\hat{\phi}(t_{ku}) = \frac{1}{n_{\xi}} \sum_{\xi=1}^{n_{\xi}} \left[t_{ku}^{\xi} - \alpha_k \right]^{+}.$$

Based on the derivation above, when there is time uncertainty, we modify the objective in (3.1) and minimize the new objective function (3.16), subject to the original constraints.

$$\begin{aligned} &\min C_a \cdot \sum_{l \in L} \sum_{k \in V} z_{lk} (\sum_{i \in M} a_{li} \cdot (1 - y_{ki})) + C_t \sum_{k \in V} \hat{\phi}\left(t_{ku}\right) \\ &= C_a \cdot \sum_{l \in L} \sum_{k \in V} z_{lk} (\sum_{i \in M} a_{li} \cdot (1 - y_{ki})) \\ &\quad + C_t \sum_{k \in V} \frac{1}{n_{\xi}} \sum_{\xi=1}^{n_{\xi}} \left[t_{ku}^{\xi} - \alpha_k \right]^{+}. \end{aligned} \tag{3.16}$$

In addition, the total tour time t_{ku} equals the sum of all traveling time and visiting time as in

$$t_{ku} = \sum_{i \in N} \sum_{j \in N} T_{kij} \cdot x_{kij} + \sum_{i \in N} T_{ki} \cdot x_{ki}, \quad \forall k \in V. \tag{3.17}$$

Suppose we know and can represent the random distributions of T_{kij} and T_{ki} as a series of samples T_{kij}^{ξ} and T_{ki}^{ξ} ($\xi = 1, \cdots, n_{\xi}$). Then, substituting (3.17) into (3.16), and linearizing the function $[x]^{+}$ using inequalities (3.19)-(3.20), the original objective function (3.1) is replaced with (3.18). Therefore, in the situation with time uncertainty, the optimization problem becomes the

following

$$\min C_a \sum_{l\in L}\sum_{k\in V} z_{lk}(\sum_{i\in M} a_{li}\cdot(1-y_{ki})) + C_t \sum_{k\in V}\frac{1}{n_\xi}\sum_{\xi=1}^{n_\xi} w_k^\xi \tag{3.18}$$

sub to $(3.2)-(3.15)$ and

$$w_k^\xi \geq \sum_{i\in N}\sum_{j\in N} T_{kij}\cdot x_{kij} + \sum_{i\in N} T_{ki}\cdot x_{ki} - \alpha_k, \quad \forall k\in V,\ \forall \xi, \tag{3.19}$$

$$w_k^\xi \geq 0, \qquad \forall k\in V,\ \forall \xi. \tag{3.20}$$

Note that w_k^ξ is a helper variable.

3.3.4 A Branch-and-Cut Approach for an Exact Solution

The optimization problem (with or without uncertainty) described in the above section is a mixed-integer quadratic program (MIQP) with bilinear objective function and linear constraints. Such a MIQP can be solved through a branch-and-cut algorithm where multiple continuous quadratic optimization problems are solved iteratively. A branch-and-cut algorithm tries to find an exact solution for an MIQP: it gives a solution and the optimality gap associated with the solution that indicates how close the solution is to the global optimum. Given enough time, such an exact algorithm will ultimately find the global optimum for a problem. Some available commercial solvers implement branch-and-cut algorithms. We encode the presented math formulation using the GUROBI solver.

3.3.5 A Large Neighborhood Search Approach

Exact algorithms are guaranteed to find the global optimum given enough time. However, for an NP-hard problem like the SMRP in this chapter, the exact optimum requires exponentially growing computations. Therefore, exact solution algorithms are usually applied to problems of smaller sizes. Here, we provide a heuristic algorithm that provides potential solutions with low objective values for larger problem cases.

According to the formulation in Sec. 3.3.2, the only nonlinear part in the optimization problem is the bilinear term in the objective function (3.1), if we fix either of the variables z_{lk} or y_{ki}, and optimize the sub-problem with the remaining variables, the optimization becomes linear. Therefore, to solve the optimization, we can fix part of the variables, limit the search space to a subset of the whole feasible domain, find the solution for a simpler linear sub-problem, and then change the search space and repeat the process to improve the solution.

The idea of solving a larger problem by iteratively solving still-large but simpler sub-problems is called a large neighborhood search (LNS). The difference between LNS and a small neighborhood search (SNS), such as the gradient descent algorithm and trust-region algorithms, is that in each iteration, the size of the search space is much larger, which ends up with larger step sizes and fewer iterations. The LNS algorithm has been widely adopted and has shown its efficiency in many scheduling and routing-related applications [123, 132–135].

We now describe how we apply an LNS algorithm to solve the SMRP problem with and without time uncertainty in Sec. 3.3.2 and 3.3.3. The pseudo-code of the large neighborhood search algorithm for SMRP is in Algorithm 2. The input of the algorithm is the unsolved optimization problem with all predefined parameters. From a set of randomly initialized variable values, the algorithm optimizes a matching sub-problem and n_V (the number of robots) routing sub-problems in each step to improve the solution quality iteratively. Note that both sub-problems are linear and with fewer variables, and therefore, much easier to solve than the original SMRP. Since the solving processes of the two sub-problems interleave in each step, the quality of the matching and routing are simultaneously optimized. The following paragraphs describe the matching sub-problem and routing sub-problem in detail.

Matching Sub-problem: Fix the variables x_{kij}, y_{ki}, t_{ki} and solve the SMRP optimization within the neighborhood of z_{lk}. Both the problems with and without uncertainty reduce to the

Algorithm 2 Large Neighborhood Search

Input: the unsolved optimization problem in Sec. 3.3.2-3.3.3
Randomly initialize x_{kij}, y_{ki}, z_{lk}, t_{ki}
for iteration $= 1, \cdots, n_{\max}$ **do**
    Fix x_{kij}, y_{ki}, t_{ki} $\quad (\forall k \in V,\ \forall i, j \in N)$
    Solve the matching sub-problem in (3.21)
    Update z_{lk} $\quad (\forall k \in V,\ \forall l \in L)$
    for $k \in V$ **do**
        Fix z_{lk} $\quad (\forall l \in L)$
        Solve the routing sub-problem in (3.23) or (3.24)
        Update x_{kij}, y_{ki}, t_{ki} $\quad (\forall i, j \in N)$
    if the objective value does not change **then**
        break
return the solution x_{kij}, y_{ki}, z_{lk}, t_{ki}

following optimization problem (variables are highlighted in blue).

$$\min \quad \sum_{l \in L} \sum_{k \in V} z_{lk} \Big(\sum_{i \in M} a_{li} \cdot (1 - y_{ki}) \Big) \tag{3.21}$$

$$\text{sub to} \quad z_{lk} \in \{0, 1\} \tag{3.22}$$

$$\text{and} \quad (3.13) - (3.15).$$

This is a smaller integer linear program (ILP) with fewer variables than the original SMRP and can be solved much faster. If there is no maximum dropped requests constraint (3.14) or human dependency constraint (3.15), then, this is a standard bipartite matching problem whose integer solutions can be obtained by solving the reduced linear program (removing constraint (3.22)) in a polynomial time [105]. We encode the math formulation here using GUROBI.

Routing Sub-problem: Fix variable z_{lk} and solve the SMRP optimization within the neighborhood of x_{kij}, y_{ki}, t_{ki}. Mathematically, the problem then reduces to a mixed-integer linear program (MILP). In terms of practical application, it becomes a variation of the vehicle routing problem.

Moreover, since the human-robot matching is fixed (fix variables z_{lk}), a robot only needs to consider the requests of the people in its own team. Therefore, the optimization is decoupled into multiple single-vehicle routing problems (i.e., traveling salesman problems). Mathematically, this

means the optimization of x_{kij}, y_{ki}, t_{ki} decouples between different robots $k \in V$. Using the large neighborhood search algorithm, we solve n_V single-vehicle routing sub-problems.

The decoupled routing problem for a single problem is shown below. For this version without uncertainty, we can leverage established routing solvers, which efficiently return low-cost solutions using multiple heuristics. Here we choose the Google Or-Tools to encode this model.

$$\begin{aligned} \min \ & C_a \sum_{l \in L} z_{lk} \left(\sum_{i \in M} a_{li} \cdot (1 - y_{ki}) \right) + C_t \, t_{ku} \\ \text{sub to } & (3.2) - (3.11). \end{aligned} \tag{3.23}$$

The sub-problem formulation with uncertain traveling and visiting times is shown below. Again, we encode the math formulation here using GUROBI.

$$\begin{aligned} \min \ & C_a \sum_{l \in L} z_{lk} (\sum_{i \in M} a_{li} \cdot (1 - y_{ki})) + \frac{C_t}{n_\xi} \sum_{\xi=1}^{n_\xi} w_k^\xi \\ \text{sub to } & (3.2) - (3.11), \text{ and } (3.19) - (3.20). \end{aligned} \tag{3.24}$$

Note that in (3.23) and (3.24), the decision variables are y_{ki}, t_{ku}, and w_k^ξ, while z_{lk} is fixed.

3.4 Local Navigation and multi-robot Simulation

In this section, we describe a simulation environment that we developed for testing the performance of the proposed framework under uncertainty. An indoor environment is used to demonstrate the possible use of the system in a museum environment. In addition, to complete the framework, we also discuss the local navigation planner and the strategy used. The simulation pipeline is implemented in ROS [136], which contains a multi-robot tour planner ROS node, multiple local planner nodes for a single robot, and a simulation wrapper node.

3.4.1 Local Navigation Planner

A local navigation planner generates the actions according to the current robot position and the routes generated by the multi-robot tour planner. Listed in Table 3.2, there are four discrete actions that a robot can take. Since localization and mapping is not the focus of this research, robot positions are assumed to be known in the simulation.

3.4.2 Habitat-AI Simulation Environment

A real-time simulation environment is used to simulate the dynamics and obstacle avoidance behavior of the robots involved in this work. The simulation is implemented in the Habitat-AI environment, a photo-realistic, physics-based simulator, to ensure that the framework and test results are transferable to real-world deployment [137–139] We choose a scene from the Matterport3D dataset, which is a collection of 90 scans, to design a sample tour of an indoor environment. Though the simulated environment is a house, it contains most of the features that would exist in a museum. The top view of the chosen scene is depicted in Fig. 3.6. And three first-person views of the robots are shown in Fig. 3.7.

3.4.3 Uncertainties in Navigation

While the planned tour from the high-level planner specifies the path that a robot should execute, the actual paths of the robots have some inherent uncertainties. The tour planner considers such uncertainties as edge traveling time (between two POIs) and node visiting time (at each POI). This section describes the factors in the simulation that introduce uncertainties to the traveling and

Table 3.2: The discrete action space for the robot.

0	STOP
1	MOVE_FORWARD (0.25 m)
2	TURN_LEFT (10°)
3	TURN_RIGHT (10°)

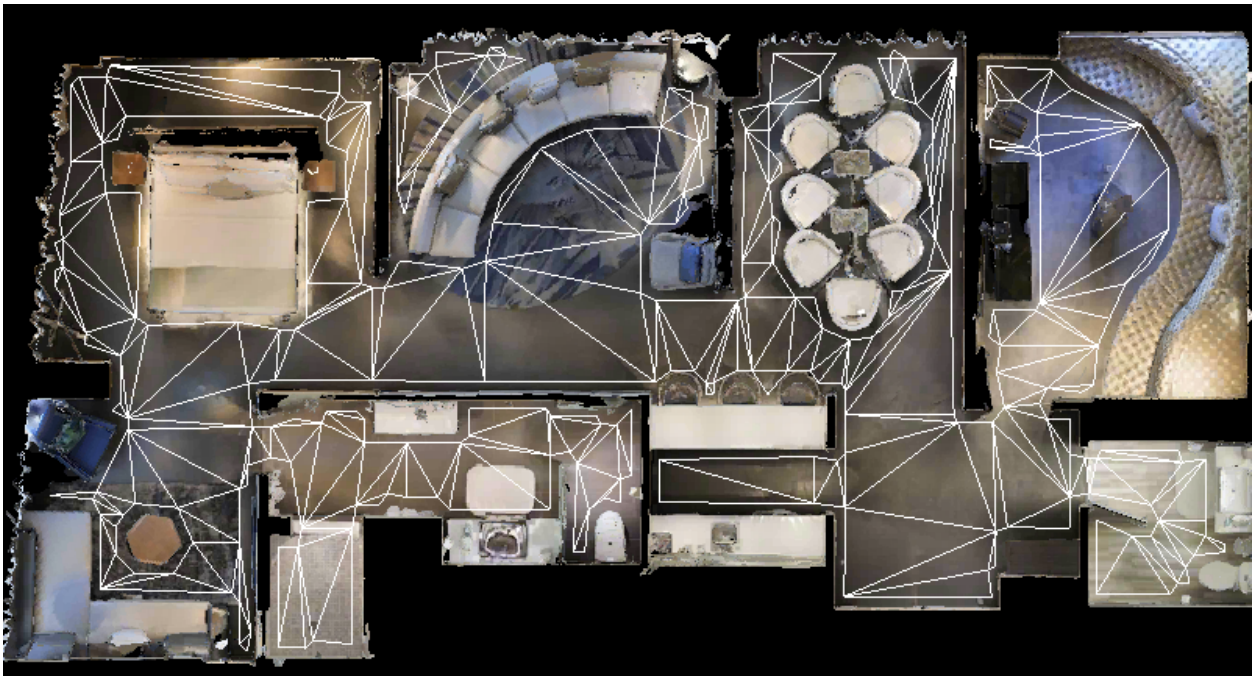


Figure 3.6: Top view of the chosen scene within the Matterport3D dataset to give a house tour within the Habitat-AI simulation environment. The white bounding boxes describe the navigable area in the scene, rendered as meshes.

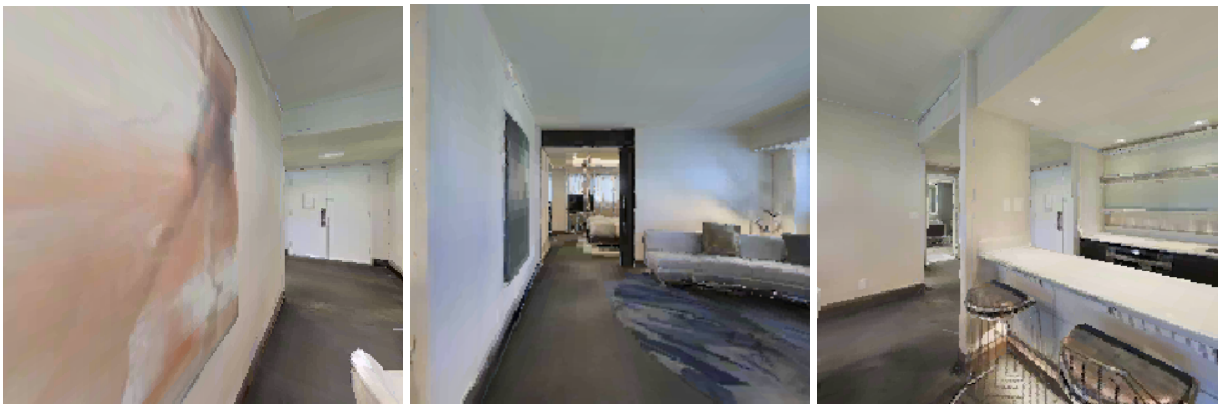


Figure 3.7: Example views of the robots conducting tasks.

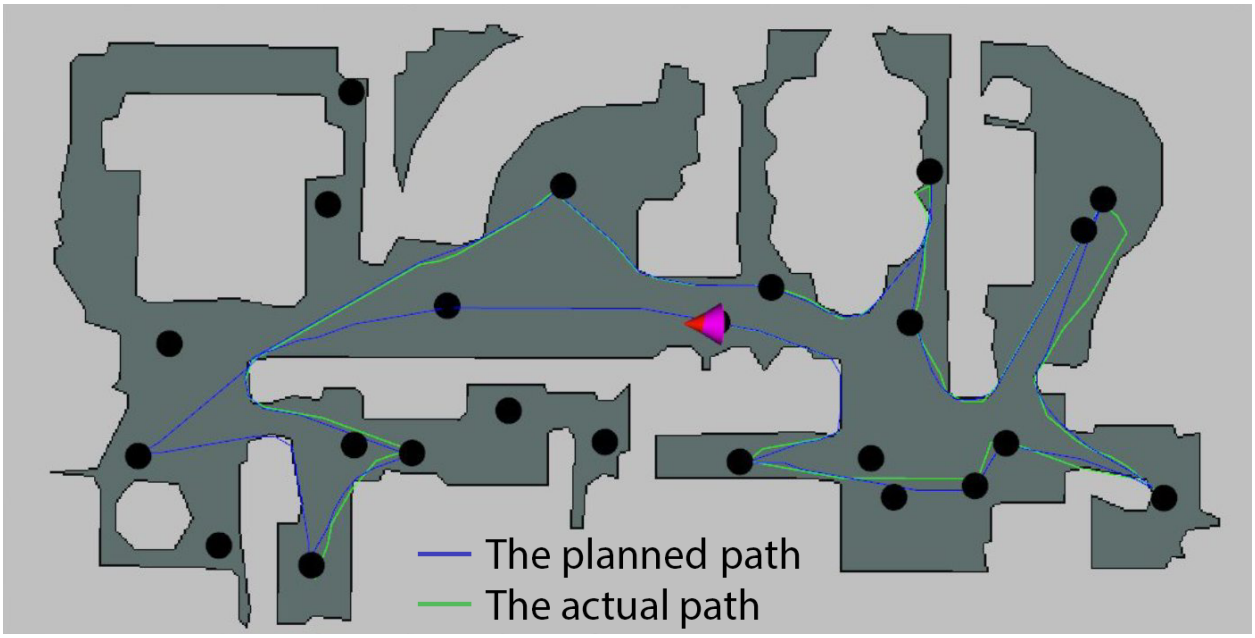


Figure 3.8: The planned path from the tour planner and the actual path in the presence of action uncertainty (10% random actions). Due to the overlap, the actual path is only visible when the robot deviates from the ideal behaviors.

visiting time.

3.4.3.1 Uncertainties in the Traveling Time

Disturbances and Imperfect Actions: Due to the lack of an accurate map and the dynamic objects in the world, the robot might need to adjust the planned tour routes according to the actual environment changes. A robot could also take wrong actions because of imperfect perception, action selection, and actuation. In the simulation, these factors are modeled as incorrect robot actions, which inject uncertainties to the edge time. The actions here are a set of movements (listed in Table 3.2) that the robots can take. For instance, if the correct action for the robot is to move forward and it actually turns left, an incorrect action happens. We add a rate of correct robot actions in the simulator to control the level of incorrectness. As an example, when this rate is set to 90%, a robot only takes the correct actions 90% of the time and random actions for the remaining 10%. The effect of 10% incorrect actions is depicted in Fig. 3.8.

Robot Dynamics: Another set of uncertainties in edge time comes from the robot dynamics.

The multi-robot tour planner estimates the traveling time according to traveling distance and average robot velocities. However, the actual dynamics of the robot system depend on the sequence of the actual robot actions and results in varying velocity and in-place rotations. The dynamics lead to variations of the actual traveling time, which is also considered a traveling time uncertainty. Note that this is an uncertainty that we are not able to control in the simulator.

3.4.3.2 Uncertainties in the Visiting Time

The interaction time is uncertain for a robot tour guide interacting with people at a POI. This is considered as a visiting time uncertainty in the multi-robot tour planner, and robust plans are generated to ensure the satisfaction of the user-specified constraints of time under uncertainty. In the simulator, the visiting time at each POI is sampled from a Gaussian distribution centered at a manually defined nominal time.

3.5 Experiments and Results

In this section, we first use randomly generated cases to evaluate the scalability and optimality of the proposed planning algorithms. These computational experiments are done on a laptop with an Apple M1 chip, and the corresponding results are shown in Sec. 3.5.1. Then we evaluated both the multi-robot tour planner in a photo-realistic simulation environment where the human's visiting time at a POI is uncertain and the robot could perform imperfect actions (which result in longer execution time). The simulation results are described in Sec. 3.5.2.

3.5.1 Computational Evaluation of the Tour Planning

We proposed two math models: a deterministic formulation and a stochastic formulation considering uncertain traveling and visiting time. We also propose two solution algorithms for the math models: an exact solution method that tries to find the global optimum and a LNS-based method that returns a heuristic solution. We will evaluate three combinations listed in Table 3.3.

Table 3.3: Algorithm types tested for the simultaneous matching and routing problem.

Abbr	Meaning
D-LNS	Deterministic formulation with LNS algorithm
D-ES	Deterministic formulation with exact solution method
S-ES	Stochastic formulation with exact solution method

Table 3.4: The ratio of the dropped requests using D-LNS. The results in boxes are where the result of D-LNS are larger (worse) than the results from the exact method D-ES, results without boxes are the opposites.

	n_V =4 n_L =10	n_V =10 n_L =50	n_V =20 n_L =100	n_V =50 n_L =250
n_M =10	0.06	0.20	0.26	0.27
n_M =20	0.00	0.26	0.31	0.45
n_M =30	0.00	0.20	0.31	0.44
n_M =50	0.03	0.12	0.28	0.36

3.5.1.1 Scalability Evaluation

The size of the problem mainly depends on the robot number n_V, human number n_L, and the total number of POIs in the area n_M. Here we variate these three hyper-parameters and test the scalability of the algorithms under the randomly generated test cases. Specifically, We choose $n_V \in \{4, 10, 20, 50\}$, $n_L \in \{10, 50, 100, 250\}$, and $n_M \in \{10, 20, 30, 50\}$. For other hyper-parameters, we choose a fixed value for all cases as they are not related to the scalability test. Particularly, for the scalability evaluation, the penalty for dropping a request, C_a, is set to 1000, while the penalty on the time, C_t, is set to 1, to ensure the dominant goal is to satisfy human requests. For each test case, we set a limit of 120 seconds for the solvers. For optimality, we use the dropped requests ratio, which is defined as (dropped requests / total requests) to indicate the solution quality.

Results show that the largest problem where D-ES and S-ES can find a non-trivial solution within 120 seconds for a problem consisting of 10 robots, 50 humans, and 50 POIs (a trivial solution is defined as dropping all requests and conducting no tour). In contrast, D-LNS can return a non-trivial solution for all the cases. For the case with 50 robots, 250 humans, and 50 POIs, the D-

LNS completes the optimization within 26.5 seconds and 7 iterations. The corresponding dropped requests ratio of D-LNS is listed in Table 3.4. According to the table, for the same deterministic SMRP, the LNS-based algorithm D-LNS outputs a better solution within 120 seconds except for the smallest cases.

3.5.1.2 Time Limit and Dropped Requests

With unlimited time, the generated matches and tours can satisfy all human requests. Therefore, the multi-robot planner tries to find the most efficient tours that drop the least number of requests within a limited tour length. We use the smallest test case ($n_V = 4, n_L = 10, n_M = 10$) as an example to show the trade-off between the tour time limit and the dropped requests ratio. Fig. 3.9 shows a smooth and close-to-linear trade-off. Therefore, during a practical situation, we can select an optimal trade-off point accordingly.

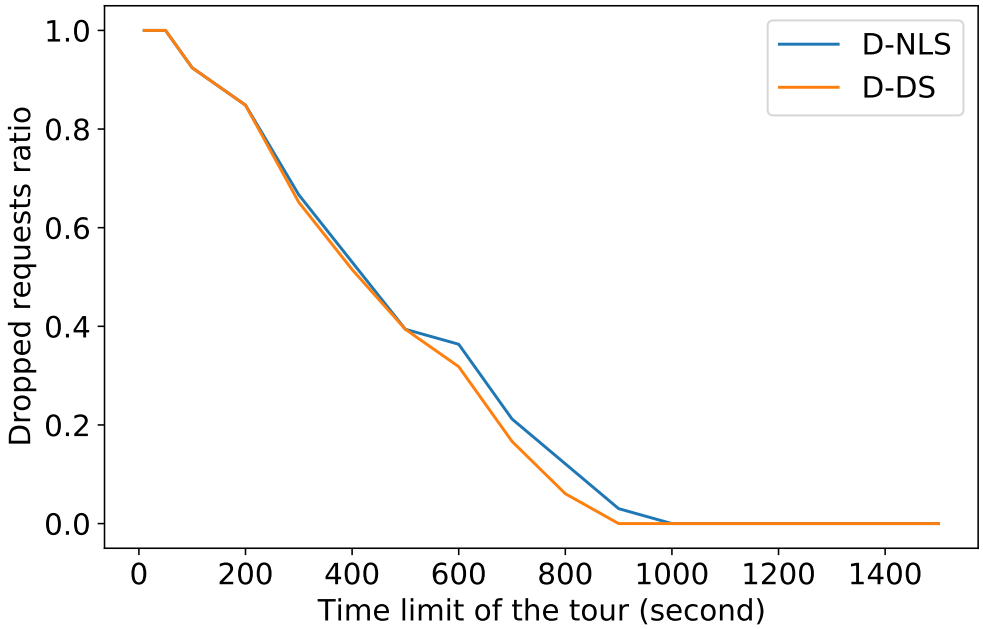


Figure 3.9: Trade-off between the tour time limit and dropped requests ratio.

3.5.1.3 Time Uncertainty and Dropped Requests

We use the smallest test case ($n_V = 4, n_L = 10, n_M = 10$) as an example to show the effect of uncertainty in an SMRP. In this parametric study, we assume the traveling time T_{kij} and visiting time T_{ki} are Gaussian distributed, and the ground truth standard deviation is 30% of the expected value, which means the robots' imperfect actions will cause the actual tour time distributed in a range specified by 30%. To apply the method S-ES to consider such uncertainty, the planner also needs to specify an estimated distribution for these time parameters. When the estimated standard distribution (the level of uncertainty) changes, the planner changes its trade-off between satisfying human requests and ensuring the tour time limit. Below, we change the standard deviation of the time distribution input to the planner and show the corresponding trade-off. The metrics used are dropped requests ratio and the average probability of surpassing the time limit during the tour.

According to Fig. 3.10 and taking into account the randomness within the optimization, when the estimated standard deviation is larger, more requests are dropped in order to decrease the probability of surpassing the time limit. When a larger uncertainty is estimated, the planner becomes more cautious about time. However, the marginal decrease in the probability becomes small when the estimated standard deviation exceeds a certain threshold. Besides, accurately estimating the uncertainty level, 30%, provides a good trade-off (both the dropped requests and the probability of surpassing the time limit are small). But a slightly off estimation (e.g., estimate the standard deviation as 20% or 40%) still provides a similarly good trade-off. This shows that the optimization is not sensitive to inaccurate uncertainty estimation.

3.5.1.4 An Example Case

Here we use the case in Sec. 3.5.1.3 with 30% estimated standard deviation to show a plan generated by solving the SMRP using the S-ES method.

The 10 people can select their POIs from 10 locations (a small case is used for visualization clarity); there are 4 robots as the tour guides. According to the optimization, the human-robot team number 0-3 contains 2, 3, 2, and 3 humans, respectively (Table 3.5). The distribution of the POIs

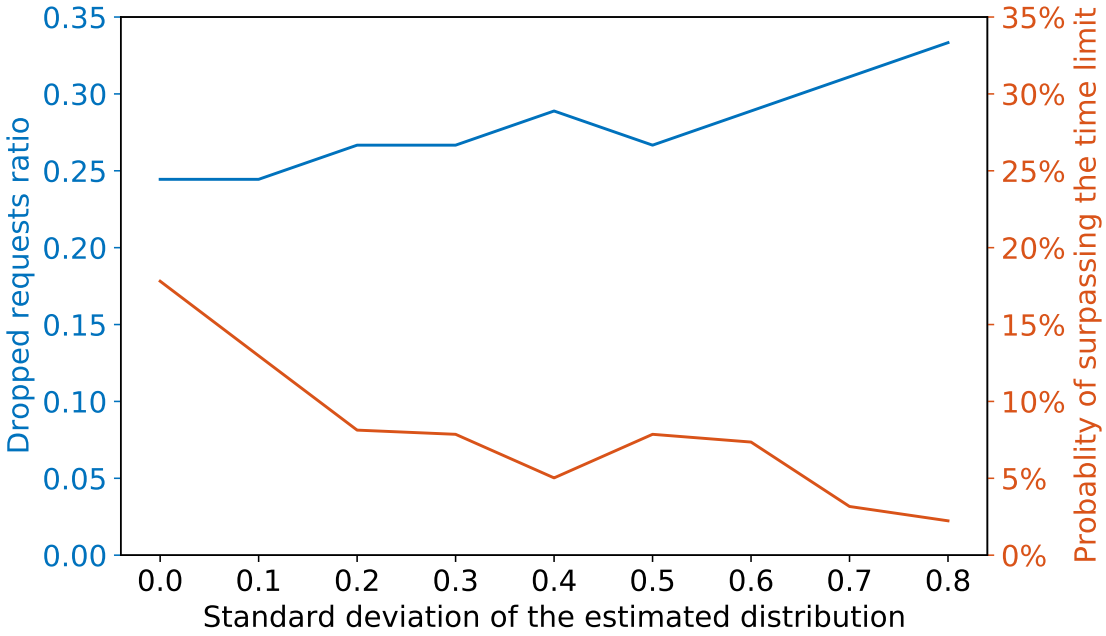


Figure 3.10: The influence of the estimated time uncertainty on dropped requests. The estimated distribution is the time distribution input to the planner. It does not necessarily equal the ground truth distribution. Here the ground truth distribution has a standard deviation of 0.3.

and the tours are shown in Fig. 3.11. Note that POI 9 is dropped by the planner in this example. The plan drops 12 of the 45 requested POIs and satisfies the rest.

Table 3.5: The planned tour and teams guided by the robots.

Robot	Tour	Time (s)	Humans
0	D, 3, 1, 2, 7, D	465	2, 8
1	D, 5, 8, 4, 0, D	517	1, 4, 9
2	D, 6, 4, 0, 1, D	522	0, 7
3	D, 3, 0, 4, 6, 8, D	426	3, 5, 6

3.5.2 Multi-robot Simulation in Habitat-AI

In this section, we run the whole framework, including the multi-robot planner that solves an SMRP, the local navigation planner, and the simulator. As described in Sec. 3.4), in the simulator, incorrect actions, non-constant velocity, rotations, as well as the varying visiting time at each POI introduce uncertainties to the traveling and visiting time. Therefore, we first show how different

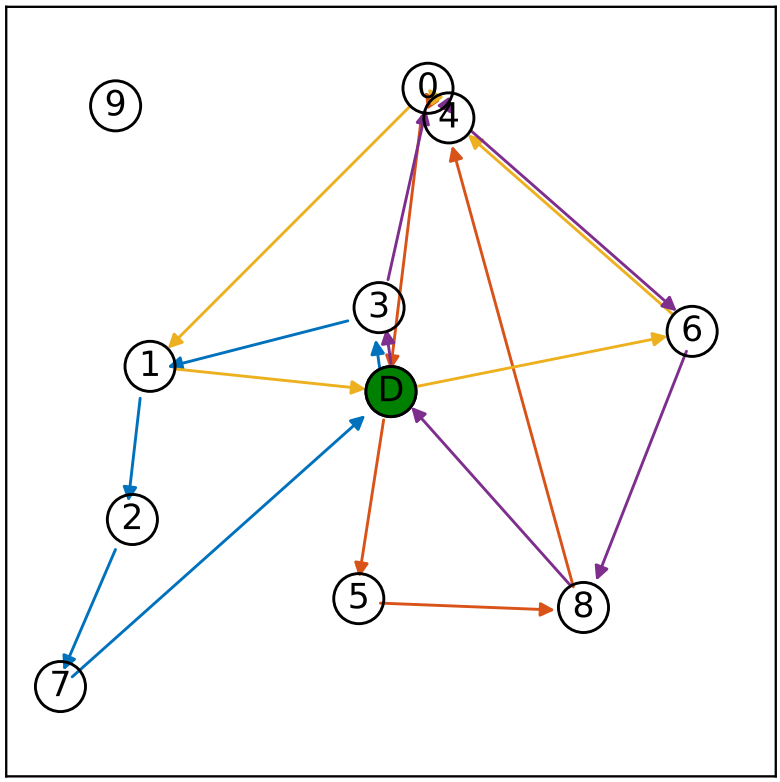


Figure 3.11: The distribution of the POIs and the routes of the robots. D refers to both the start and terminal. Robot 0: blue, robot 1: brick red, robot 2: yellow, robot 3: purple.

levels of simulated uncertainties will affect the touring time of the human-robot teams.

The SMRP planner assumes estimated distributions of time costs are available. In reality, we can gather samples to estimate the time uncertainty, but the estimation can be inaccurate. We did a parametric study on the level of environmental and estimated uncertainties to see how planned tours and actual tour times change accordingly. In the simulation, we constrain the estimated uncertainty to be Gaussian distributions, but our SMRP model with time uncertainty does not assume a specific type of random distribution.

3.5.2.1 Varying Environmental Uncertainty

We fix the plan generated by an SMRP solver with uncertainty (the assumed standard deviation is 40% of the nominated value), and for the uncertainties in the simulation, we change the rate of correct actions from (100% to 70%). Then, we show the results of the actual tour time. Since the

result is stochastic, we repeat the test five times for each setup. Note that there are 3 robots, 10 humans, and 22 POI in total for visiting in this simulation. The time limit of the tour is set to 100 seconds.

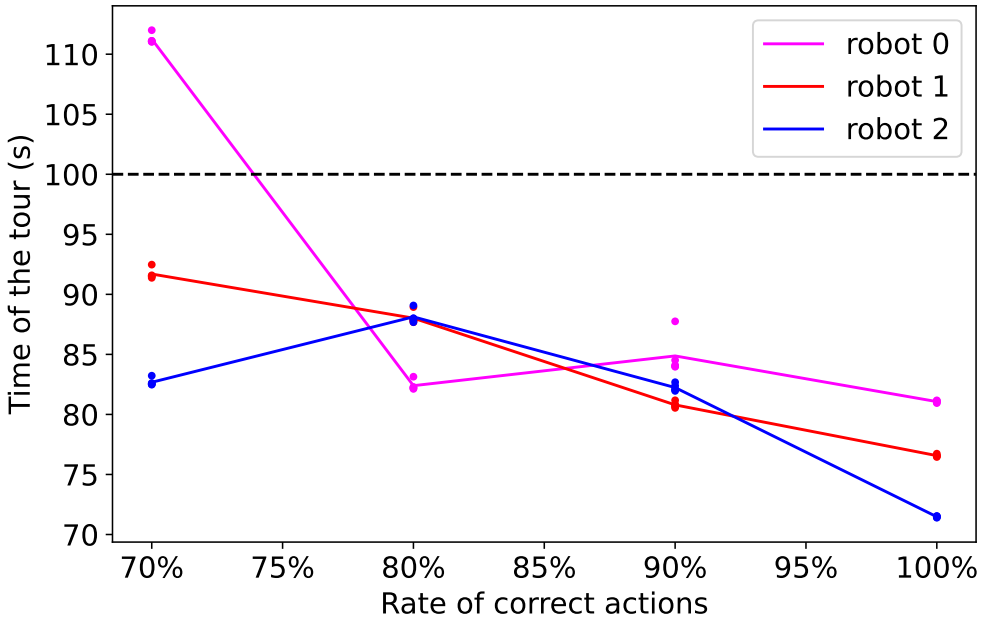


Figure 3.12: The actual tour time of the three robots when the environmental uncertainty changes. The dots show the results of multiple random trials, while the line shows the mean. The expected tour times, according to the planner, are 83, 82, and 75 seconds, respectively.

According to Fig. 3.12, with the touring routes fixed (Fig. 3.13), the tour time generally increases when the rate of correct actions decreases. Apart from that, for the situations with $\geq 80\%$ accuracy, the generated plans are always completed within the time limit.

We note that the time uncertainty is assumed as Gaussian distribution, but the actual time uncertainty introduced by incorrect actions may not necessarily be Gaussian. In addition, the standard deviation of the actual time distribution might not be 40%. Nevertheless, a fixed and possibly inaccurate uncertainty estimation (40%) can ensure the time constraints of the tour under different correct rates (80% to 100%). This shows that the generated plan has a relatively wide margin with respect to the uncertainties during the tasks. Moreover, the robust planning framework does not require a perfectly accurate uncertainty estimation to work.

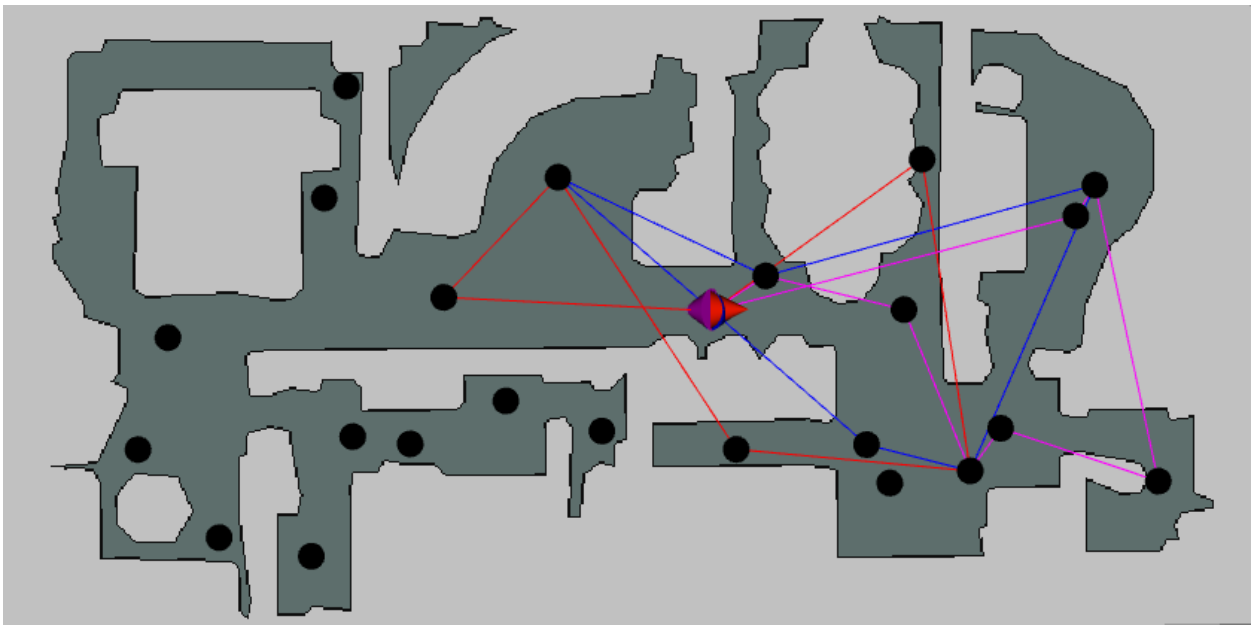


Figure 3.13: The planned tour of the three robots assuming the standard deviation is 40%. Between two POIs, straight lines instead of the actual trajectories are shown for clarity.

3.5.2.2 Varying Estimated Uncertainty

In this subsection, we fix the rate of correct actions in the simulation and vary the estimated uncertainty of the planner. We input Gaussian distributions to the planner and set the uncertainty for traveling between two POIs a fraction (0% to 100%) of the nominal value (proportional to traveling distance). Again, under one setup, the tests are repeated five times. The actual tour times are shown in Fig. 3.14. According to the figure, there is no clear trend when the standard deviation of the estimated distribution increases. For this specific case, all the plans that consider the uncertainties end up not surpassing the time limit. Again, this is a sign that a perfect estimation of the uncertainty parameters is not a hard requirement for the multi-robot SMRP planner model with uncertainty.

Finally, the planned tour with 0% and 40% standard deviations are shown in Fig. 3.15 and Fig. 3.13, respectively. And it can be seen that the tours in Fig. 3.15 are indeed longer and, therefore, more aggressive. Here, a more aggressive tour tries to address more human requests but is more likely to surpass the tour time limit.

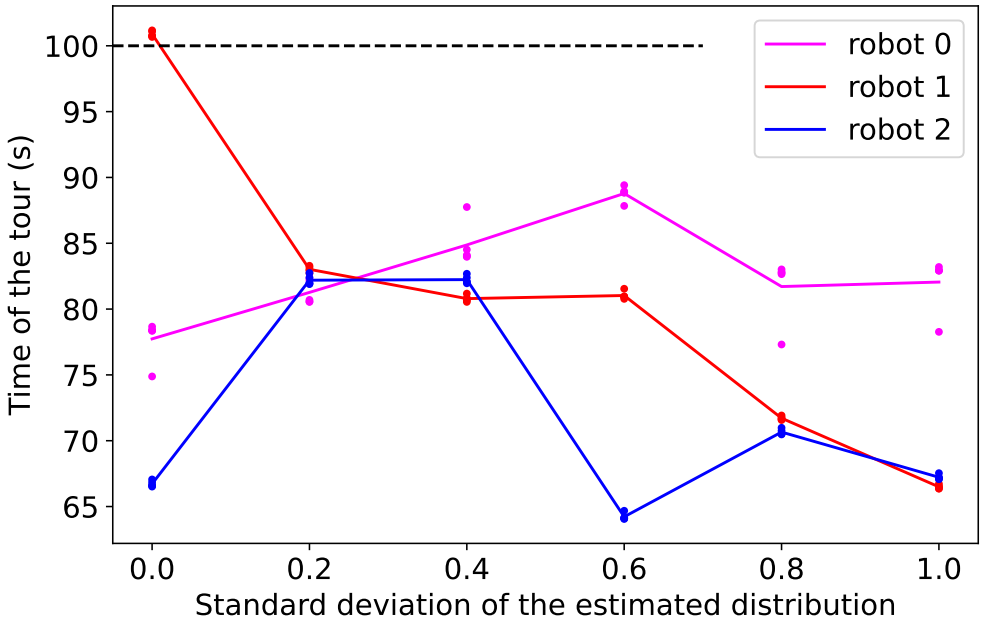


Figure 3.14: The actual tour time of the three robots when the estimated uncertainty changes. The dots show the results of multiple random trials, while the line shows the mean.

3.6 Conclusions and Future Work

This chapter presents a multi-robot framework that deals with robotic tour guiding in a partially known environment with uncertain traveling and visiting times. A two-level structure is described in the chapter: a centralized multi-robot planner generates the tours of the robots and assigns the humans to robots to maximize the number of satisfied human requests (places of interest); a distributed local navigation planner decides the actual trajectories for a single robot according to the global tour routing plan. The problem considered in the multi-robot planner is generalized and modeled as a simultaneous matching and routing problem, which simultaneously optimizes the human-robot matching and robot routing problem. The uncertainties in the traveling and visiting times are considered, and robust plans are generated by penalizing the expected tour times that pass a predefined threshold. The mixed-integer bilinear program that represents the above planning problem is solved through both an exact branch and cut algorithm as well as an innovative large neighborhood search method. The algorithms are evaluated through comprehensive computational investigations. Then, the robustness of the whole framework is evaluated in a photo-realistic

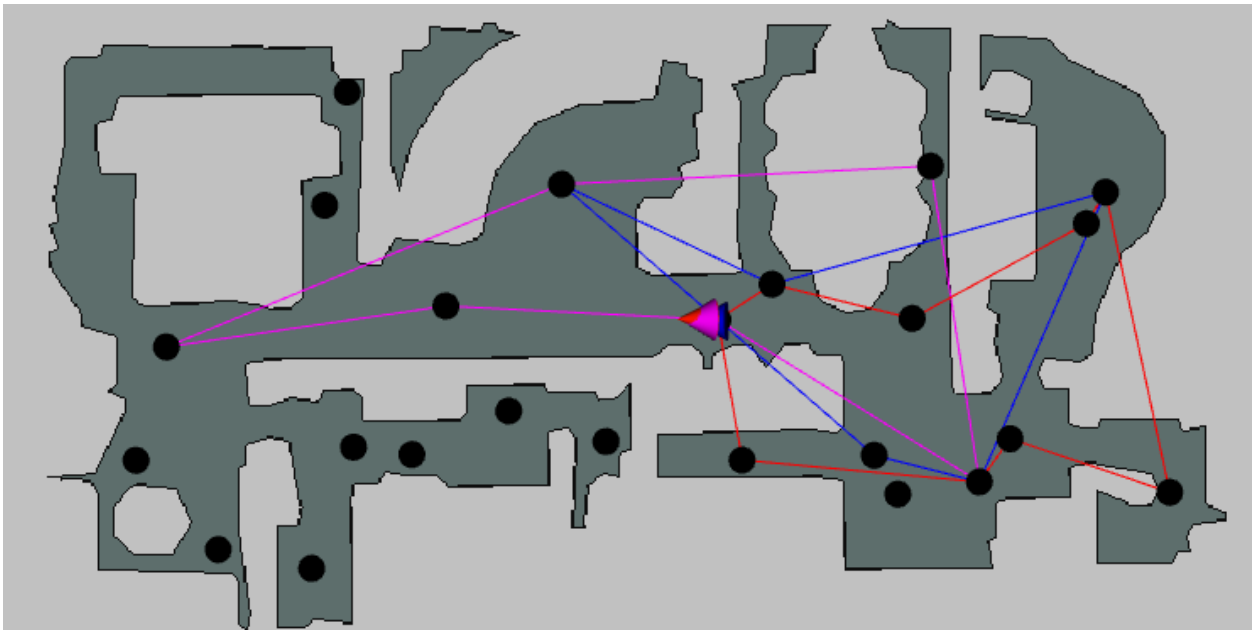


Figure 3.15: The planned tour of the three robots, assuming no uncertainty (standard deviation is 0%).

simulation that captures multiple practical uncertainties.

Results demonstrate that, through the large neighborhood search, the planner is scalable to the number of robots, humans, and POIs and can output plans with low dropped requests. The largest case tested involves 50 robots, 250 humans, and 50 POIs. Second, results show that the planner makes a smooth trade-off under the time limit for the tours, which provides valuable information for tour design. Finally, parametric study and simulation evaluation demonstrate that the planner can generate robust tours that keep the time constraints in practice with crude uncertainty estimations.

Future work will consider dynamic replanning mechanisms to address real-time disturbances or failures, as well as a more consolidated local planner that can deal with local disturbances. These are options to further improve the robustness of the tour guiding. Heterogeneity within the robots will be considered too. Examples include different sensors and actuators that allow different robots to perform distinct gestures and movements during the guidance. Real-world experiments will be a further step to demonstrate the practicality of our proposed system.

CHAPTER 4

Learning Task Requirements and Agent Capabilities for Multi-agent Task Allocation

The multi-agent task allocation framework proposed in chapter 2 can be applied to generate task assignments, schedules, and routes for the agent teams. However, it requires the task requirements and agent capabilities to be known. In fact, most optimization-based task allocation frameworks similarly assume their model parameters provided and are limited by this assumption [30, 44, 45, 56, 57, 81, 94, 140]. When model parameters are unknown or contain significant uncertainty, we need an estimation method to learn their values from data or observations.

This chapter presents a learning framework to estimate an agent capability and task requirement model for multi-agent task allocation to address contribution 2 in Fig. 1.1. With a set of team configurations and the corresponding task performances as the training data, a capability-based task requirement model used in many existing optimization-based task allocation frameworks can be learned. Comprehensive computational evaluations are conducted to test the scalability and prediction accuracy of the learning framework with a limited number of team configurations and performance pairs. A ROS and Gazebo-based simulation environment is developed to validate the proposed requirements learning model with an optimization-based task allocation in practical multi-agent exploration and manipulation tasks. Results show that the learning process for scenarios with 40 tasks and 6 types of agents uses around 13 seconds, ending up with prediction errors in the range of 0.5-2%.

The rest of the chapter is organized as follows: Sec. 4.1 motivates the research topic to learn

the modeling parameters for optimization-based task allocation frameworks. Sec. 4.2 briefly summarizes the related work of this chapter. Sec. 4.3 introduces the learning model that estimates the task requirements and agent capabilities. Sec. 4.4 leverages and simplifies a model from [45] to illustrate how the learned task requirements can be applied in a task allocation model. Sec. 4.5 presents the computational experiments that evaluate the prediction accuracy, computation time, and scalability of the learning model. Sec. 4.6 describes a simulation case study that applies the framework in Fig. 4.2. Sec. 4.7 concludes the chapter with ideas for current limitations and future work.

This chapter is adapted from our paper [141].

Supplementary videos and open-source code for this project can be found at:
https://brg.engin.umich.edu/publications/learn-multiagent-taskreq.

4.1 Introduction

Multi-agent systems have been applied to achieve complex cooperative robotic tasks[13, 16, 24, 29, 50, 95]. Consider the task allocation problem in Fig. 4.1. There are three types of agents (v_2, v_4, v_5) with different exploration and manipulation capabilities and five types of tasks that require different agent teams to complete. A planner should determine an appropriate team for each task and generate consistent routes and schedules for all agents to optimize time and energy usage while subject to task requirements, routing constraints, and other practical constraints. For example, in Fig. 4.1, one agent of v_2, two agents of v_4, and three agents of v_5 together complete the five tasks in the sequences and routes illustrated in the figure.

Such a task allocation problem with heterogeneous agents and tasks has been considered in optimization-based or reinforcement learning frameworks [30, 44, 45, 81, 94, 140, 142, 143]. While optimization model-based systems usually require no training and provide predictable performance in new scenarios, they often assume the model parameters (e.g., task requirements) are known or can be provided [30, 44, 45, 81, 94, 140]. However, in practice, model parameters are

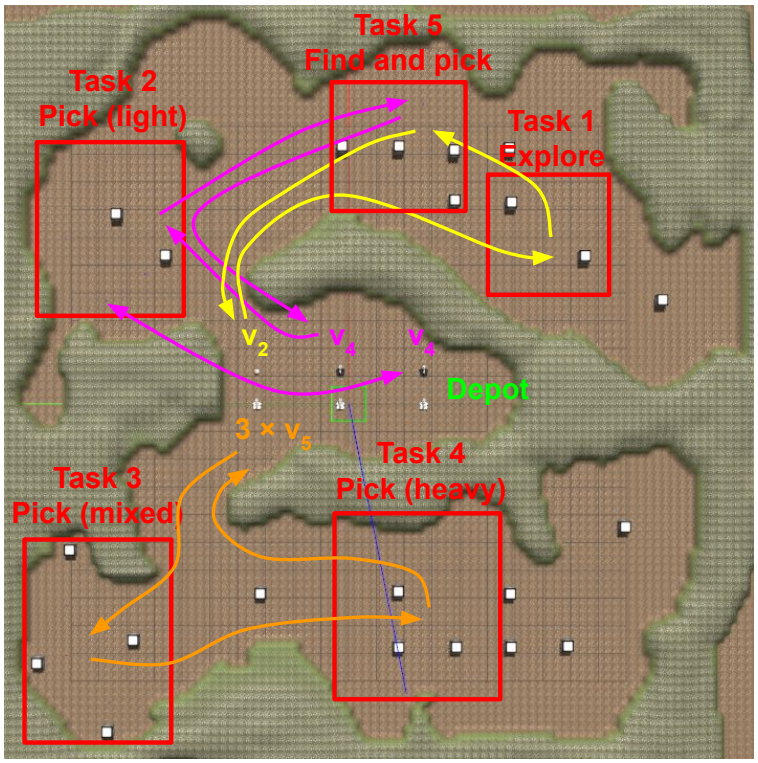


Figure 4.1: A multi-agent task allocation problem consisting of five heterogeneous tasks. An agent from type i is marked as v_i in the figure. The teams for the tasks are shown as follows. Task 1: v_2, task 2: two v_4, task 3 and 4: three v_5, task 5: one v_2 and one v_4.

often unknown or contain significant uncertainty. Therefore, most existing optimization-based approaches are limited by this assumption. To address the limitation, this work presents a method to learn the parameters of a task requirement and agent capability model, which is widely used in optimization-based task allocation [30, 44, 45, 81, 94]. The capability-based model assumes that agents possess sets of discrete capabilities and that tasks require specific quantities of a given capability to successfully complete an objective. The model can be embedded as linear constraints in the optimization.

The system architecture is summarized in Fig. 4.2. The right block shows a typical optimization-based task allocation model [30, 44, 45, 81, 94] that generates allocation plans based on the agent capabilities and task requirements, parameterized by A^* and $\mathbf{b}^*$. This part is leveraged from existing work. The middle block describes the main contribution of this chapter - a learning model that estimates the parameters (A^* and $\mathbf{b}^*$) of the capability-based model. The left

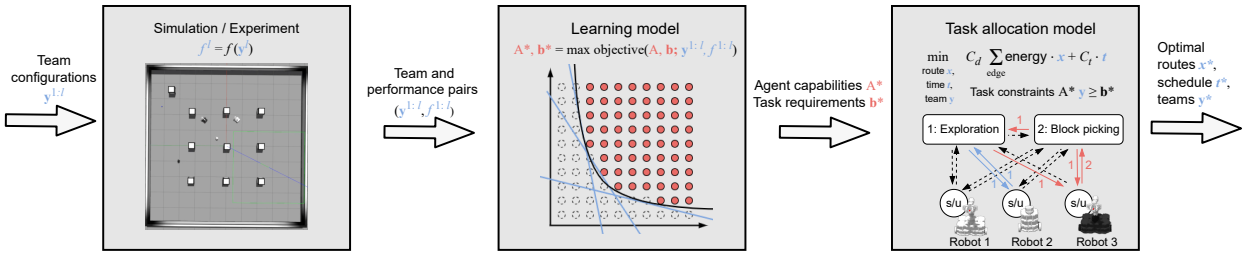


Figure 4.2: System architecture. In the graph for the task allocation model, there are two task nodes, and s/u means the start and terminal nodes for the three agent types. The edges are highlighted if there is a number (also highlighted) of agents traveling the edge.

block is a simulation/experiment environment for evaluating team configurations and their task performances, which are the training data of the learning model. The fitted task constraints can be considered a function that returns 1 if a team is valid. Once the parameters A and $\mathbf{b}_i$ are learned, the function can evaluate the validity of an unseen team configuration $\mathbf{y}_i$.

The procedure can be shown to be applied to different types of tasks (generalizable) when the assumption of a cumulative and independent capability and known sparsity pattern, provided in Sec. 4.3, is satisfied. Note that the sparsity pattern defines whether an entry in the agent capabilities A^* and task requirements $\mathbf{b}^*$ is zero/nonzero.

This chapter provides the following contributions.

1) The development of a generalizable framework with training data gathering, task requirement learning, and task allocation planning for multi-agent tasks.

2) The formulation of a linear program that learns the parameters of a task requirement and agent capability model for task allocation.

3) Comparison against a baseline algorithm for learning the task requirements for task allocation.

4) Implementation of a ROS and Gazebo-based multi-agent simulation that validates the proposed framework through a practical case study. Open-source code: https://brg.engin.umich.edu/publications/learn-multiagent-taskreq/

4.2 Related Work

4.2.1 Optimization model-based Multi-agent Task Allocation

Both learning-based [38, 144–147] and optimization model-based frameworks [30, 44, 45, 81, 94, 142, 143] have been applied to the task allocation problem. Learning-based frameworks usually achieve high performance in similar test environments. However, they often require a large amount of training data as they leverage little prior knowledge from the problem structure.

Optimization frameworks model the relationship between the tasks and agents. In [66, 74], a task is represented as a graph of states (nodes) and actions (edges) so that graph search algorithms can find the best action sequences for agents. In [142, 143], the task constraints are specified using linear temporal logic. The works of [44, 45] apply the capability model described in Sec. 4.1 and solve a discrete stochastic optimization to obtain the routes, schedules, and task assignment plans. The works of [30, 81, 94] utilize a similar capability-based model but formulate the problem as an optimal control-style continuous optimization to generate the desired capability distribution at each time step.

4.2.2 Learning Optimization Models from Data

There are many examples from previous works focused on learning the parameters in an optimization model for single-robot manipulation and navigation. The work of [66] leverages a learning-from-demonstration approach to determine a graph of actions for a new manipulation task. Authors of [148, 149] develop methods to learn temporal logic from demonstrations for robot navigation and manipulation.

Multi-agent task allocation, on the other hand, has fewer previous related works. As mentioned in Sec. 4.2.1, the works of [30, 44, 45, 81, 94] start from the agent capability model and represent the task requirements as a vector of thresholds on each capability type. The authors of [30, 81, 94] present a method to estimate task-required capability values in [140]. They assume agent capabilities are known and estimate task requirements based on appropriate team configurations.

However, an agent's task capabilities can be hard to determine, and assuming this knowledge a priori limits the application of the framework. The framework we present in this work enables simultaneous learning of agent capability values and task requirement thresholds. In Secs. 4.5-4.6, we use the method in [140] as a baseline and compare the prediction errors and task allocation performances for this baseline method versus our proposed framework.

4.3 Learning Task Requirements and Agent Capabilities as Linear Constraints

The system architecture is shown in Fig. 4.2. Suppose the task performance is a function of the team configuration (given a task environment) that can be evaluated through a simulation or experiment. In real-world applications, this value can be task completion time, product outputs, or a combined quantity. With team configurations and performance pairs, the learning model tries to learn the parameters for the capability-based task requirement model used in [30, 44, 45, 81, 94]. We propose the use of a capability-based task model that we formulate as a set of linear task constraints embedded in an optimization-based task allocation. The following section reformulates the learning of the agent capabilities and task requirements as an optimization problem with learning linear constraints.

4.3.1 Task Performance Function

Consider a set of agent types $K = \{1, \cdots, |K|\}$, and tasks $M = \{1, \cdots, |M|\}$. Suppose y_{ki} is the number of agents with type $k \in K$ assigned to a task $i \in M$. And $\mathbf{y}_i = [y_{1i}, \cdots, y_{|K|i}]^\mathsf{T} \in \mathbb{Z}_{\geq 0}^{|K|}$ denotes the number of different agents at task $i \in M$. Let the task performance be a function of the team configuration: $f_i(\mathbf{y}_i) : \mathbb{Z}_{\geq 0}^{|K|} \to \mathbb{R}$ for each task $i \in M$. Note that $\mathbb{R}$ and $\mathbb{Z}_{\geq 0}$ are the set of real numbers and non-negative integers, respectively. Ideally, we should set a lower bound, f_i^*, on

the performance of a given task as follows

$$f_i(\mathbf{y}_i) \geq f_i^*, \quad \forall i \in M, \tag{4.1}$$

and embed it in the task allocation optimization. Depending on the performance metric used, the $\geq$ can be replaced with $\leq$. However, $f_i(\mathbf{y}_i)$ is evaluated through an experiment or simulation. It is non-analytic and cannot be directly added as a constraint in the optimization. In this section, a set of linear constraints is learned to approximate (4.1).

4.3.2 Task Requirements and Agent Capabilities

This subsection summarizes the capability-based task requirement model used in [30, 44, 45, 81, 94]. Consider a set of agent capability types $C = \{1, \cdots, |C|\}$ that can be used to characterize an agent's ability to perform a task (e.g., perception, manipulation, or transportation). Suppose a_{ck} represents the value of capability $c \in C$ that agent type $k \in K$ possesses. We concatenate these capability values into a capability matrix (see Fig. 4.3 for example)

$$\begin{aligned} A &= \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_{|K|} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{a}^1 \\ \vdots \\ \mathbf{a}^{|C|} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1|K|} \\ \vdots & \ddots & \vdots \\ a_{|C|1} & \cdots & a_{|C||K|} \end{bmatrix} \end{aligned}$$

where $\mathbf{a}_k = [a_{1k}, \cdots, a_{|C|k}]^\mathsf{T} \in \mathbb{R}_{\geq 0}^{|C|}$ is a vector indicating the capabilities of agent type $k \in K$.

Task requirements for a given task $i \in M$ are provided as a vector of thresholds $\mathbf{b}_i = [b_{1i}, \cdots, b_{|C|i}]^\mathsf{T}$, where b_{ci} denotes the threshold on capability type $c \in C$.

For simplicity, the capabilities are assumed cumulative.

Assumption 1 (Cumulative Capability)**.** *If multiple agents work together, the total capability of the team is the sum of their individual capability values.*

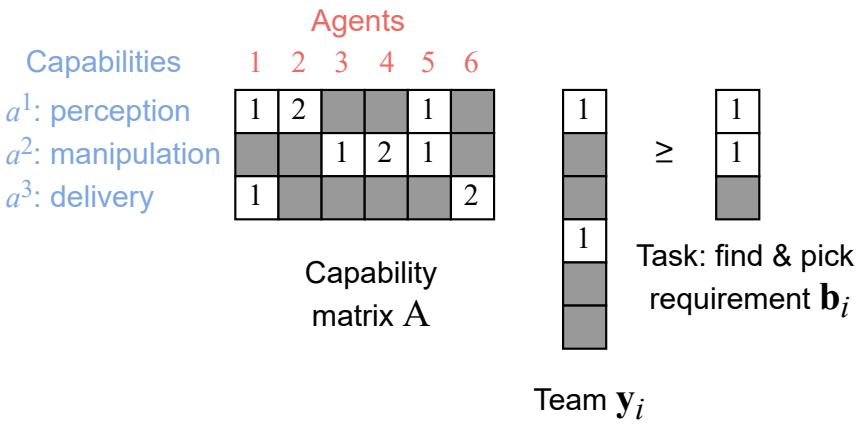


Figure 4.3: An example team formation problem using a capability matrix and task requirement vectors. Agents can perform three capabilities: perception, manipulation, and delivery. The example task requires perception to explore and locate an object and manipulation to grasp the object.

Thus, a team's capability set for a given task can be calculated via matrix multiplication $A\mathbf{y}_i \in \mathbb{R}^{|C|}$. Future work will relax this assumption to allow for non-cumulative capabilities. The capabilities are assumed independent.

Assumption 2 (Independent Capability)**.** *The constraints of different capabilities are independently placed.*

With the two assumptions leveraged from [30, 44, 45, 81, 94], the task requirement constraints are placed as follows

$$A\mathbf{y}_i \geq \mathbf{b}_i, \quad \forall i \in M. \tag{4.2}$$

Equation (4.2) can be illustrated graphically as in Fig. 4.3. From the figure, we can see that one capability (a row in A and $\mathbf{b}_i$) forms one linear constraint, and multiple linear constraints together form the task requirements.

4.3.3 Learning the Task Requirements and Agent Capabilities

With the task requirement model above and the cumulative capability assumption, the actual task constraints in (4.1) are approximated using linear constraints in (4.2). Ideally, the two con-

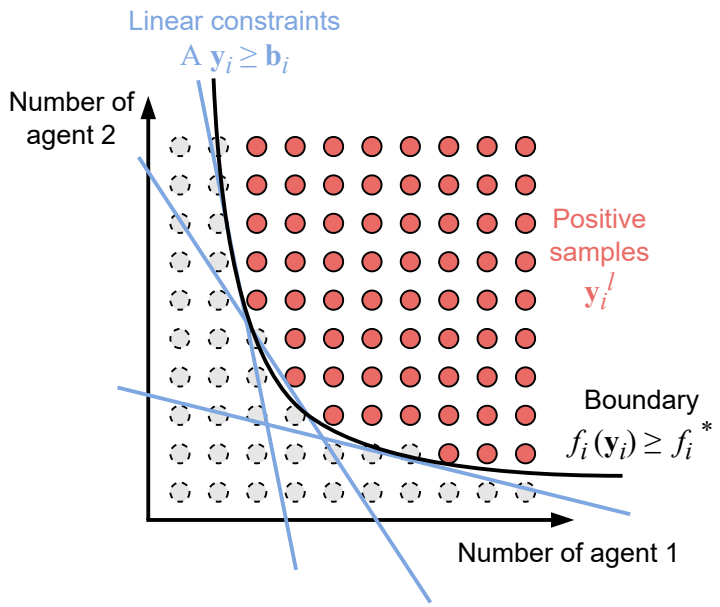


Figure 4.4: Finding the tightest linear boundaries for the positive samples.

straints should correspond to each other as follows

$$A\mathbf{y}_i \geq \mathbf{b}_i \Leftrightarrow f_i(\mathbf{y}_i) \geq f_i^*, \quad \forall i \in M. \tag{4.3}$$

However, as demonstrated in Fig. 4.4, linear constraints can only describe convex polytopes. Thus, instead of a one-to-one mapping as stated in (4.3), we aim to fit the tightest linear boundaries for the region $f_i(\mathbf{y}_i) \geq f_i^*$. Recall that $f_i(\cdot)$ is non-analytic; thus, the region of $f_i(\mathbf{y}_i) \geq f_i^*$ is not explicitly available. To compensate for this, we evaluate the performance of different team configurations, among which $\mathbf{y}_i^l, (l = 1, \cdots, n_L)$ are shown to satisfy $f_i(\mathbf{y}_i^l) \geq f_i^*$ (denoted as positive samples in Fig. 4.4). We then fit linear boundaries to these positive samples, and use the following linear program to find the A and $\mathbf{b}_i$ associated with the tightest linear boundaries.

$$\begin{aligned} &\max_{A,\mathbf{b}_i} \sum_{i \in M} ||\mathbf{b}_i||_1 \\ &\text{sub to } A\mathbf{y}_i^l \geq \mathbf{b}_i, \ \ \forall l = 1, \cdots, n_L, \ \ \forall i \in M, \\ &\quad ||\mathbf{a}^c||_1 = 1, \ \forall c \in C. \end{aligned} \tag{4.4}$$

In (4.4), the objective function maximizes the requirements thresholds $\mathbf{b}_i$ to find the tightest value. Note that $||\cdot||_1$ stands for the L_1 norm. The first constraint ensures that all of the positive samples satisfy the constraints defined by A and $\mathbf{b}_i$. For example, if we scale the capability matrix A and requirements vector $\mathbf{b}_i$ by the same value, the constraints $A\mathbf{y}_i \geq \mathbf{b}_i$ do not change. Given that the scale of the capabilities is unconstrained, we add the second constraint in (4.4) to normalize each type of capability.

The optimization problem in (4.4) can further be written (row-wise) into the following form.

$$\max_{\mathbf{a}^c, b_{ci}} \sum_{c\in C}\sum_{i\in M} \mathrm{abs}(b_{ci}) \tag{4.5}$$
$$(\mathbf{a}^c)^\mathsf{T}\mathbf{y}_i^l \geq b_{ci},\ \ \forall l = 1, \cdots, n_L,\ \ \forall i \in M,\ \ \forall c \in C,$$
$$||\mathbf{a}^c||_1 = 1,\ \forall c \in C.$$

Note that $\mathrm{abs}(\cdot)$ denotes the absolute value. Based on (4.5) and assumption 2, the optimization of $\mathbf{a}^c$ and b_{ci} can be decoupled for different capabilities $c \in C$. This is also illustrated in Fig. 4.3, since different types of capabilities do not appear together in a single linear constraint and thus do not affect each other. Therefore, we can decompose the optimization presented in (4.5) into $|C|$ separate linear programs with an additional term.

$$\max_{a_{ck}, b_{ci}} \left(\frac{\alpha_b}{|M|} \sum_{i\in M} b_{ci} + \alpha_a \min_{k\in K, (c,k)\in\mathcal{A}_1} a_{ck} \right) \tag{4.6}$$
$$\sum_{k\in K} y_{ki}^l\, a_{ck} \geq b_{ci},\ \ \forall l = 1, \cdots, n_L,\ \ \forall i \in M, \tag{4.7}$$
$$\sum_{k\in K} a_{ck} = 1, \tag{4.8}$$
$$a_{ck} > 0, \quad \forall (c,k) \in \mathcal{A}_1, \quad a_{ck} = 0, \quad \forall (c,k) \in \mathcal{A}_0, \tag{4.9}$$
$$b_{ci} > 0, \quad \forall (c,i) \in \mathcal{B}_1, \qquad b_{ci} = 0, \quad \forall (c,i) \in \mathcal{B}_0. \tag{4.10}$$

The two constraints in (4.5) are rewritten in (4.7)-(4.8). The objective function in (4.5) is refor-

mulated in (4.6) with an additional penalty. Consider the general case (an example is given in Fig. 4.3), for which we assume that we know the type of agent capabilities and task requirements, but do not know the specific values for these classes[1]. We summarize this knowledge in the following assumption:

Assumption 3 (Sparsity Pattern)**.** *We assume a priori knowledge about whether an entry in A and $\mathbf{b}_i$ is positive (e.g. exists) or zero (e.g. does not exist).*

Suppose $\mathcal{A}_1 = \{(c,k)|a_{c,k} > 0\}$ and $\mathcal{A}_0 = \{(c,k)|a_{c,k} = 0\}$ are the sets of index pairs for positive and zero agent capabilities, and $\mathcal{B}_1 = \{(c,i)|b_{c,i} > 0\}$ and $\mathcal{B}_0 = \{(c,i)|b_{c,i} = 0\}$ are the index pairs for positive and zero task requirements, respectively. We express this prior knowledge in constraints (4.9)-(4.10). The right term in (4.6) is added to ensure that positive capabilities are indeed learned to be positive. Leveraging the prior knowledge of the sparsity pattern slightly improves the estimation accuracy. The selection of the weights (α_b, α_a) was determined through a parametric study, where the results indicated that the solutions are not sensitive to the weight values. In general, choosing $\alpha_a = 0.05\alpha_b$ would work for most cases with less than 32 capability types.

As a summary, given a set of team configurations and the corresponding task performances $(\mathbf{y}_i^l, f_i(\mathbf{y}_i^l)), (l = 1, \cdots, n_L)$, with prior knowledge about whether an entry in the agent capability A or task requirement $\mathbf{b}_i$ is non-zero, we can solve $|C|$ small linear programs (with $|K| + |M|$ variables) to estimate the linear task requirement constraints for task allocation. As linear programming is polynomially solvable, the learning model developed here is scalable to the number of tasks, agents, and training samples.

4.3.4 Motivation and Effect of the Assumptions

Three assumptions are made during the formulation of the learning model. The cumulative and independent capability assumptions together linearize the task requirements. Therefore, the

[1]For example, we may assume that if a robot contains a manipulator, that it has the potential to perform a manipulation task (hence contains a manipulation capability). However, we do not assume knowledge about the value of this capability (e.g. weight or size constraints).

requirements can be represented as linear constraints in (4.2), simplifying the parameter learning process. However, it requires the shape of the valid team configuration to be a convex polytope. This requirement is generally satisfied for tasks that can be distributed to multiple agents and being completed in parallel. The model cannot be applied to non-cumulative capabilities (e.g., the total speed a team can traverse is not determined by the sum of the individual agent speeds). The model cannot be applied to non-convex task requirements where, for example, some task requirements are optional if others are satisfied.

The sparsity pattern assumption is added to leverage prior knowledge about an agent's capabilities (as it is usually available) to reduce the training data size and improve parameter estimation accuracy. If no sparsity pattern is given, there can be formulations to estimate it from the data.

4.4 Task Allocation with Learned Requirements and Agent Capabilities

Many task allocation frameworks represent the task constraints using a capability-based model [44, 45, 81, 94]. This section leverages an optimization model from [45] (a simplified version) to show how the learned constraints in Sec. 4.3 can be embedded in a task allocation problem. We use centralized task allocation as an example; however, the learned capability-based model can also be embedded in a distributed system.

Consider the set of tasks, $M = \{1, \cdots, |M|\}$, distributed at different locations, each requiring a team with the correct capabilities to complete the tasks. A set of agent types $K = \{1, \cdots, |K|\}$ are located at the depot. The goal is to complete the tasks using a team of available agents while minimizing the weighted sum of the time and energy spent to travel to and complete the tasks.

4.4.1 Graphical Model

We construct a graph $G(N, E)$ with the node set $N = \{1, \cdots, |M|, s, u\}$ and edge set $E = \{(i, j)|\forall i, j \in N\}$ (Fig. 4.5). There are edges between every node pair. The node set contains the

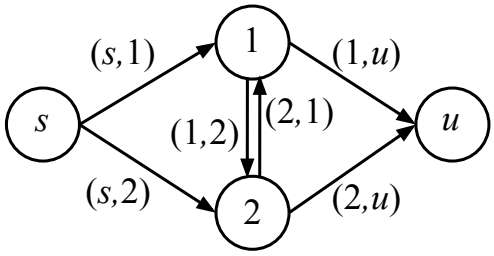


Figure 4.5: A graphical model example with $|M| = 2$. I.e., there are two task nodes, a start s, and a terminal u.

tasks 1 to $|M|$, start $s = |M| + 1$, and terminal nodes $u = |M| + 2$. Note that the integer set M is a subset of N. A practical example of this graphical model is shown in the right-most block of Fig. 4.2. In this graph, the agents depart from the start node, visit all of the tasks assigned to them, and stop at the terminal node.

4.4.2 A Mixed-integer Program for Task Allocation

In this section, we formulate a mixed-integer program for the task allocation problem based on the graphical model in Fig. 4.5. Common notations are explained in Table 4.1. Note that some of the notations have already been defined in Sec. 4.3, but are still gathered here for clarity.

4.4.2.1 Objective Function

Minimize the energy cost and the time to complete all the tasks, subject to the stated constraints. C_d and C_t are the weights. The decision variables are x, y, r, and t.

$$\min_{x_{kij}, y_{ki}, r_{kij}, t_i} C_d \sum_{k \in K} \sum_{i,j \in N} d_{kij} \cdot x_{kij} + C_t \, t_u \tag{4.11}$$

4.4.2.2 Task Requirement Constraints

The task requirement constraints are learned according to Sec. 4.3.

$$A\mathbf{y}_i \geq \mathbf{b}_i, \quad \forall i \in M. \tag{4.2}$$

Table 4.1: Definition of the notations.

	Meaning
x_{kij}	The quantity of agent traveling on edge $(i, j) \in E$ with species $k \in K$.
y_{ki}	The quantity of agent $k \in K$ assigned to task $i \in M$.
$\mathbf{y}_i$	= $[y_{1i}, \cdots, y_{\|K\|i}]^\mathsf{T} \in \mathbb{Z}_{\geq 0}^{\|K\|}$, the number of agents at different tasks.
r_{kij}	= 1 if $x_{kij} \geq 1$, otherwise 0. This is a helper variable indicating whether an agent is traveling an edge.
t_i	The time task $i \in M$ starts.
t_u	The time all the agents arrive at their terminals.
T_{kij}	The time for agent k to travel edge (i, j).
T_{ki}	The time for agent k to complete its role at task $i \in M$.
T_{large}	A predetermined large time constant.
d_{kij}	The energy cost for agent k to travel edge (i, j).
D_k	The energy limit for agent k.
A	The learned agent capability matrix.
$\mathbf{b}_i$	The learned task requirement threshold for task $i \in M$.

4.4.2.3 Helper-variable Constraints

The following constraint relates the x and r variables: $r_{kij} = 1$ if $x_{kij} \geq 1$ and $r_{kij} = 0$ if $x_{kij} = 0$. r_{kij} indicates whether there are agents from type k traveling edge (i, j).

$$x_{kij} \geq r_{kij}, \quad x_{kij} \leq n_k \cdot r_{kij}, \quad \forall i, j \in N, \forall k \in K. \tag{4.12}$$

4.4.2.4 Time Constraints

The time constraints ensure: first, enough time for the agent to travel the edges from task i to task j; second, all agents arrive before the next task j starts.

$$t_i - t_j + T_{kij} + T_{ki} \leq T_{\text{large}}(1 - r_{kij}), \forall i, j \in N, \forall k \in K. \tag{4.13}$$

4.4.2.5 Energy Constraints

The sum of the energy cost of an agent should be smaller than its stated limit.

$$\sum_{i,j \in N} d_{kij} \cdot x_{kij} \leq D_k, \quad \forall k \in K. \tag{4.14}$$

4.4.2.6 Network Flow Constraints

(4.15) ensures that the agents entering a task also leave after the task is completed. (4.16) sets an upper bound on the number of agents from each type. n_k is the number of available agents from type k. (4.17) relates the x and y variables: the agent team at a task is comprised of the agents that arrive at the task from all of the edges.

$$\sum_{i \in N} x_{kim} = \sum_{j \in N} x_{kmj}, \quad \forall m \in M, \ \forall k \in K. \tag{4.15}$$

$$\sum_{i \in M} x_{ksi} \leq n_k, \qquad \forall k \in K. \tag{4.16}$$

$$y_{kj} = \sum_{i \in N} x_{kij}, \quad \forall j \in M, \ \forall k \in K. \tag{4.17}$$

4.4.2.7 Variable Bounds

x, y, and t are continuous variables, while r variables are binary.

$$\begin{aligned} &x_{kij} \geq 0, \ y_{ki} \geq 0, \ r_{kij} \in \{0,1\}, \forall i,j \in N, \forall k \in K. \\ &t_i \geq 0, \quad \forall i \in N, \quad t_s = 0. \end{aligned} \tag{4.18}$$

Once a solution to the optimization problem is obtained, the variables encode the information about the schedule (t), routes (x), and teams (y) for the tasks. As an example, in the right-most block in Fig. 4.2, the edges are colored and numbered according to the values of the x variables. According to the solution, two Robot 3 first complete the block-picking task, and then one of Robot 3 drives to the exploration task and completes it together with one Robot 2.

4.5 Computational Evaluation

In this section, we focus on the computation example to evaluate the scalability and optimality of the learning model. All computations are done on a laptop with an Apple M1 chip. We compare the results with a baseline algorithm [140]. Note that we will show an example of applying our framework to a practical problem in Sec. 4.6.

4.5.1 Test Cases

Assume for a set of tasks with performance functions $f_i(\cdot), i \in M$, that the assumption in (4.3) holds; i.e., the allocation of teams to accomplish the given task can be formulated as a set of linear constraints. We use the following $f_i(\mathbf{y}_i)$ to generate the performance of a team $\mathbf{y}_i$ during the computation evaluation (f_i^* in (4.3) is set to 0.5 for all task i).

$$f_i(\mathbf{y}_i) = \begin{cases} 1, & A^g \mathbf{y}_i \geq \mathbf{b}_i^g \\ 0, & \text{otherwise} \end{cases} \quad \forall i \in M. \tag{4.19}$$

A^g and $\mathbf{b}_i^g$ are preset ground truth capabilities and task requirements, and are assumed to be unknown to the learning model. The goal of the learning model is to estimate A and $\mathbf{b}_i$ based on the $f_i(\mathbf{y}_i)$ samples.

We define eight test cases of different sizes (given in Table 4.2) to test the accuracy and computational time of the learning model. For each case in Table 4.2, we randomly initialize 10 realizations with different ground truth matrices A^g and $\mathbf{b}_i^g$ in (4.19) and generate the data $(\mathbf{y}_i^l, f_i(\mathbf{y}_i^l)), l = 1, \cdots, n_L$. For a particular task, the team configurations exist within $\mathbf{y}_i \in [0, n_s]^{|K|}$, where n_s is the number of agents within each type. As an example, if $n_s = 5$ and there are $|K| = 6$ related agent types, and the size of the configuration space is $46656 = (5+1)^6$.

Table 4.2: Test cases of different sizes and the training times. $|M|$ is the task number. $|K|$ is the number of agent types. n_s is the number of agents within each type. $|C|$ is the number of capabilities. $\bar{n}_L$ is the mean size of the team configuration space for the tasks, while $\sum n_L$ is the sum. Entire: is the average training time using the entire team configuration space. Random: is the average training time using a random subset of team configurations.

Case	$\|M\|$	$\|K\|$	$\|C\|$	n_s	$\bar{n}_L$	$\sum n_L$	Entire	Random
0	8	6	8	5	1.5k	12k	6.1	0.6
1	8	6	8	5	6k	48k	21.6	0.6
2	8	6	16	5	6k	48k	53.1	1.1
3	8	6	32	5	6k	48k	109.4	2.7
4	20	6	8	5	7.5k	150k	62.1	1.5
5	40	6	8	5	7.5k	300k	108.5	2.8
6	40	6	16	5	7.5k	300k	262.0	6.3
7	40	6	32	5	7.5k	300k	570.4	12.6

4.5.2 Accuracy and Computational Cost

Our proposed learning model and the baseline are used to estimate the capabilities and task requirements. The learned values of A and $\mathbf{b}_i$ are then used to predict whether a team can complete a task according to (4.19). The results are compared to the label generated by the ground truth A^g and $\mathbf{b}_i^g$. Since the baseline algorithm in [140] requires a priori knowledge of the agent capabilities, we input the capabilities A learned by our algorithm to the baseline model to estimate the task requirements $\mathbf{b}_i$.

For a real-world problem, obtaining team performance samples can be expensive (through experiments/simulations). To investigate whether the model learns with a small training set, we evaluate the algorithm in two modes: 1) training and testing both on the entire team configuration space (up to 46656 samples for a task); 2) training on at most 200 randomly selected team configurations for each task while testing across the entire configuration space.

Considering the ideal case expressed in (4.3), and given a set of test data $(\mathbf{y}_i^l, f_i(\mathbf{y}_i^l)), l = 1, \cdots, n_L$, we define the prediction error (approximation error) for task i as follows

$$e_i = \sum_l \text{abs}\left(\mathbf{1}_{f_i(\mathbf{y}_i^l)\geq f_i^*}(\mathbf{y}_i^l) - \mathbf{1}_{A\mathbf{y}_i^l\geq \mathbf{b}_i}(\mathbf{y}_i^l)\right)/n_L, \tag{4.20}$$

where $\mathbf{1}_E(\cdot)$ is the indicator function that returns 1 if the expression E is satisfied and, otherwise, 0.

The distribution of mean prediction errors of all cases using our model and the baseline is shown in Fig. 4.6. Our algorithm can accurately recover the capability and task requirements with prediction errors ranging from 0.5-2%. The test errors from the results trained on the subset (200 random samples) are slightly larger than the model trained on the entire data set. The baseline algorithm leads to larger prediction errors in all cases. To understand the impact of the training set, we conducted a parametric study on the number of random samples used in the training. This study found that the prediction errors gradually increased and were smaller than 2% when using more than 50 random samples (0.6% of the complete configuration space). Additional samples will be needed for a system that contains more capability types. Applying the use of a random team selection strategy results in low prediction errors partially due to the implicit enforcement of diversity in the team configurations. Therefore, the task requirements and agent capabilities can be estimated using a small percentage of the data within the configuration space.

Recall that our proposed learning model assumes that the sparsity pattern is known. However, we also evaluated scenarios when we have inaccurate sparsity knowledge. Although the prediction errors increase under this uncertainty, the capabilities can still be estimated. For example, a 10% error in the sparsity matrix resulted in prediction errors on the order of 2-6% for the eight cases, compared to the original range from 0.5-2%.

The average training times are listed in Table 4.2. The computational times grow linearly with respect to the number of tasks and capabilities. For case 7 (the largest case), the average training times using the entire configuration space versus the randomly selected subsets are 570 and 13 seconds, respectively. From this analysis, one can determine that the learning model can easily scale to a mission comprised of 40 tasks and 32 different capabilities if a subset of tasks is used for learning the capabilities and task requirements. From the times provided in Table 4.2, one can determine that the model can update the agent capabilities and task requirement values every few seconds for real-time applications.

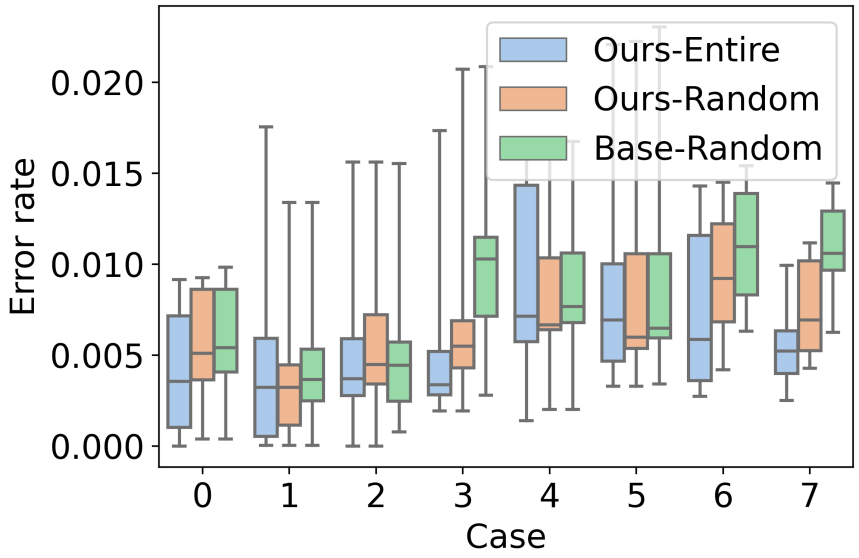


Figure 4.6: Prediction errors tested on the complete data, compared to the baseline. 'Entire' refers to the results estimated using the entire configuration space (up to 46656 teams), while 'Random' refers to the results estimated from a subset of the configurations. 'Base' refers to the baseline.

4.6 Simulation Example

This section illustrates our learning framework using a robotic exploration and manipulation problem. Multiple types of tasks are involved in the example to show that the framework generalizes to different tasks.

The simulation environment is developed in Gazebo, and the communication between robots is established using ROS. A screenshot is shown in Fig. 4.7. There are five types of agents, and their qualitative capabilities are described in Table 4.3. There are five types of tasks as shown in Table 4.4. The indoor environment in Fig. 4.7 is used to gather team performance samples to estimate the agent capabilities and task requirements. Then, we apply the optimization-based task allocation in Sec. 4.4 to generate teams, routes, and schedules for the outdoor task setup in Fig. 4.1. Once the team configuration is determined for a specific task, a simple intra-team strategy (greedy algorithms) will guide the agents to complete the task cooperatively. As this chapter focuses on inter-team scheduling and team configuration planning, the simple intra-team strategy is not described here.

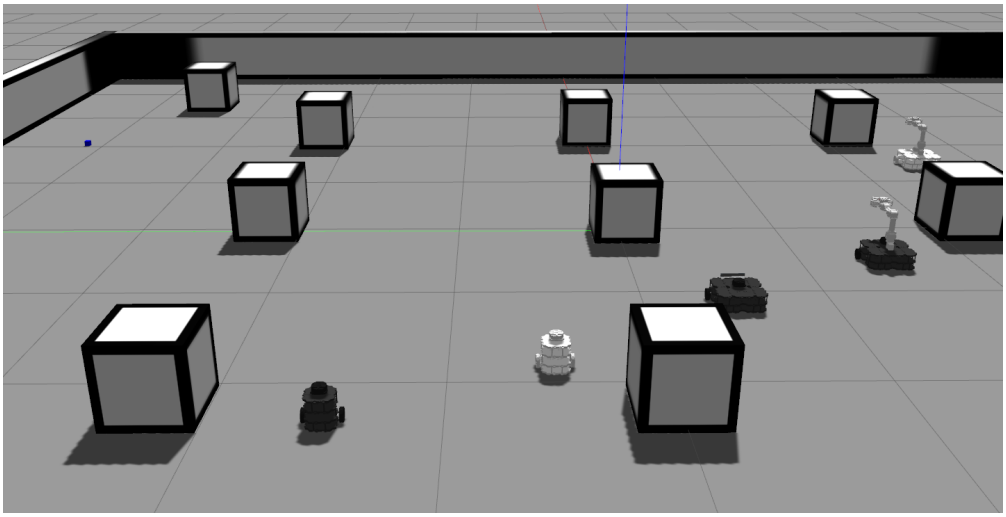


Figure 4.7: A screenshot of the multi-agent simulation environment. The robot team is conducting a 'find and pick' task.

Table 4.3: Qualitative agent capabilities. A screenshot of the five robots is shown in Fig. 4.7 (left to right: agent type 1-5).

Agents	Perception	Manipulation
1: Smallbot 1	The range is 1 m.	None.
2: Smallbot 2	The range is 1.5 m.	None.
3: Largebot 1	The range is 1 m.	None.
4: Largebot 2	The range is 1 m.	Only picks light blocks. Picks light blocks faster than largebot 3.
5: Largebot 3	The range is 1.5 m.	Picks both light and heavy blocks.

4.6.1 Learning the Capability Model

Taking into consideration the physical size of the task area, we limit the team size for a given task to four agents. We chose to evaluate the performance of 20, 14, 10, 4, and 25 team configurations for the five tasks. The number of samples used to learn a capability or requirement scales according to the number of task-relevant capability requirements or the expected size of the teams. For example, task 1 only requires perception capabilities, while task 5 requires both perception and manipulation capabilities. As such, task 5 might need additional samples in order to learn the capability requirements accurately.

For each task $i = 1, \cdots, 5$ and team configuration $\mathbf{y}_i^l$ (l is the label for a team configuration), de-

Table 4.4: Qualitative task requirements. Perception is not required for the picking tasks as the block locations are known. The size of a task region is set to 6×6 meters.

Tasks	Perception	Manipulation
1: Explore	Cover the region.	None.
2: Pick (light)	None.	Pick four light blocks at known locations.
3: Pick (mixed)	None.	Pick two light and two heavy blocks at known locations.
4: Pick (heavy)	None.	Pick four heavy blocks at known locations.
5: Find and pick	Find a block.	Pick the found light block.

fine the task performance as $f_i(\mathbf{y}_i)$. In this case study, we consider task completion time as a critical performance objective. Note that $f_i(\mathbf{y}_i)$ is stochastic because the completion time is influenced by the localization, control, communication uncertainties, and environmental setup. Therefore, rather than solving for $f_i(\mathbf{y}_i) \leq f_i^*$, we use (4.21) to determine whether a team configuration is valid.

$$P(f_i(\mathbf{y}_i) \leq f_i^*) \geq 80\% \tag{4.21}$$

To check the condition in (4.21), we randomly generate five cases for each (i, l) pair (with different block locations) and evaluate the task completion time. If at least four of the task completion times are smaller than the task time threshold f_i^*, the team configuration is considered valid. The value of 80% can be modified to according to the specific case. The approximation accuracy will be higher if the boundary defined by f_i^* in Fig. 4.4 is convex or near convex. For a real-world problem, choosing an f_i^* that leads to balanced positive and negative samples will result in better accuracy. The task completion times are listed in Table 4.5.

The capability-based task requirements model is used to describe the problem scope for the team planning. We define five capability types (1: perception, 2: manipulation related to light blocks, 3: manipulation related to heavy blocks, 4: perception 2, 5: manipulation 2 related to light blocks). Note that we have two capability features related to the perception and manipulation of light blocks. A specific capability of an agent (e.g., perception) can change regarding different

tasks. For instance, in Table 4.6, perception 1 and 2 are relevant to tasks 1 and 5, respectively. According to Table 4.3, Smallbot2 has a larger perception range than Smallbot1, and therefore, it has a larger capability of perception 1. However, in task 5, the difference in their perception ranges is not significant enough to change the task performance, and therefore, the learned values for perception 2 are the same. Adding multiple perception-related capabilities regarding different tasks makes the capability-based model more expressive and may reduce prediction errors.

Applying our learning framework, the learned values with the thresholds listed in Table 4.5 are shown in Tables 4.6-4.7. We mentioned that capabilities are normalized as in (4.4). For simplicity of notation, we globally scale the capabilities in the tables such that the smallest value is 1. The values indicate that 1) smallbot2 and largebot3 have larger perception capabilities and 2) largebot 2 has a larger manipulation capability when picking light blocks. This is consistent with their qualitative capabilities defined in Table 4.3. The prediction error with the selected threshold is shown in Table 4.5. A parametric study has been done and shown that similar capability and task requirement values can be learned with perturbations of ± 50 over the selected threshold.

In summary, the learning model can be applied to a practical problem to learn meaningful capabilities and task requirements with low prediction errors. Further, the learning process is not sensitive to threshold selections.

Table 4.5: Task completion time and threshold selection for the tasks. The unit for the {min, max, mean, thres} are seconds. A specific team configuration $\mathbf{y}_i$ is considered valid if the completion time $f_i(\mathbf{y}_i)$ is smaller than the threshold. The error rate calculates the proportion of falsely predicted samples using the learning model.

Tasks	Sample	Min	Max	Mean	Thres	Error
1: Explore	20	85	245	156	190	0%
2: Pick (light)	14	44	211	99	110	0%
3: Pick (mixed)	10	66	252	117	130	0%
4: Pick (heavy)	4	70	203	140	140	0%
5: Find and pick	25	44	111	74	90	4%

Table 4.6: Agent capabilities A^{T} learned by our model.

Agents	1: P	2: ML	3: MH	4: P2	5: ML2
1: Smallbot 1	1	0	0	1	0
2: Smallbot 2	2	0	0	1	0
3: Largebot 1	1	0	0	1	0
4: Largebot 2	1	2	0	1	1
5: Largebot 3	2	1	1	1	1

Table 4.7: Task requirements learned by our model.

Tasks	1: P	2: ML	3: MH	4: P2	5: ML2
1: Explore	2	0	0	0	0
2: Pick (light)	0	4	0	0	0
3: Pick (mixed)	0	3	1	0	0
4: Pick (heavy)	0	0	3	0	0
5: Find and pick	0	0	0	2	1

4.6.2 Task Allocation with the Learned Capabilities

In Fig. 4.1, five heterogeneous tasks (chosen from Table 4.4) are distributed in a 30×30 meters outdoor area. We assume four agents are available from each type in Table 4.3. A task will be completed within the required time once a team with enough capabilities reaches the task region.

Applying the task allocation model in Sec. 4.4 with the use of the learned agent capabilities and requirements presented in Tables 4.6-4.7, a teaming plan can be generated to minimize a weighted sum of the energy and time while satisfying all task requirements. The planned routes and team configurations are shown in Fig. 4.1 and Table 4.8. Under the task constraints learned by our learning model, the planner chooses one agent of type 2, two agents of type 4, and three agents of type 5 to complete the tasks.

The teaming and routing plan in Fig. 4.1 is evaluated in simulation. The total time for agents to complete all tasks and travel back to the depot is 317 seconds. The task time thresholds in Table 4.5 are all maintained, indicating that the linear task constraints are an appropriate approximation of the real-world task completion time constraints.

We also apply the task allocation with the task requirements learned using the baseline method

Table 4.8: Task teams according to the learned task requirements.

	Our team	Baseline team
1: Explore	v_2	v_2
2: Pick (light)	$2 \times v_4$	$2 \times v_4$
3: Pick (mixed)	$\mathbf{3} \times v_5$	$\mathbf{2} \times v_5$
4: Pick (heavy)	$3 \times v_5$	$3 \times v_5$
5: Find and pick	v_2, v_4	v_2, v_4
Total task time	317 sec	403 sec

in [140]. Their method requires a prior knowledge of the agent capabilities. Using the capability values estimated by our learning model as the a priori knowledge of agent capabilities, we find that although the task requirements learned using the baseline method are different, the task allocation results (e.g. team configurations and performance metrics) are the same. However, if we modify the agent capabilities to reflect alternative values that may have some uncertainty, the learned task requirements using the method proposed in [140] can lead to degenerated task performances. In one of the cases, shown in Table 4.8, only two v_5 are assigned to task 3, which delays the task completion time, and hence, the total completion time increased to 403 seconds versus the original 317 seconds. This indicates the importance of simultaneously learning agent capabilities and task requirements.

Overall, the results show that, with a small amount of training data from the simulation and a few seconds of optimization time, the proposed task requirement learning is able to inform the task allocation planner to solve practical problems and generate task allocation plans. Compared to the baseline algorithm, our model does not require prior knowledge of the agent capability values.

4.7 Conclusions and Future Work

4.7.1 Conclusions

This chapter presents a learning model to estimate task requirement constraints for optimization-based multi-agent task allocation (e.g., the framework in Chapter 2). The estimation

problem is formulated as a linear program that optimally fits agent capabilities and task requirements to team configurations and task performance. For situations where the team configuration space is large and an exhaustive performance evaluation is impractical, a randomly chosen subset provides sufficient information for the estimation. Computational evaluations show that the learning model can learn the task requirements for cases with up to 40 tasks, 6 agent types, and 32 capability types within 13 seconds and with low prediction error (around 1%).

A Gazebo-based simulation platform is developed, and the learning framework is validated in a practical multi-agent exploration and manipulation problem. The learned agent capabilities and task requirements reflect the qualitative agent capabilities and task requirements in Tables 4.3-4.4. For example, larger perception ranges are reflected by the larger perception capabilities in Table 4.6. Finally, an optimization leverages the learned requirements and successfully generates task allocation plans for a simulation case study.

4.7.2 Limitations and Future Work

Despite the promising results from the simulated case study, a few limitations of our proposed framework must be stated. First, because of the assumptions made in Sec. 4.3, the model cannot be applied to non-cumulative capabilities or non-convex task requirements. Additional modeling architectures can be used to relax these assumptions within the task allocation problem. Second, the performance of a given team with respect to a specific task will typically vary due to disturbances and uncertainties in the system. These distributions should be considered within the overall framework. Future work will explore alternative learning models that encode and estimate such uncertainties within the agents' task capabilities.

CHAPTER 5

Learning-based Intra-team Trajectory Planners for Tightly Cooperative Tasks

If every single agent knows when and where to perform a single-agent action, and there are no dependencies between the single-agent activities, then a multi-agent task can be decomposed and assigned to individual agents and be completed by leveraging existing single-agent perception, planning, and control methods. The task allocation framework developed in Chapters 2-4 addresses the problem of task decomposition, assignment, and scheduling. Therefore, it fills one major gap in linking single agents and multi-agent systems.

In a loosely coupled system, the trajectory of one agent performing its actions does not impact the trajectory of another. As a result, their individual activities can be conducted independently. For instance, in vehicle routing in large open spaces [45, 120–122], it is unlikely that two vehicles will collide following their assigned routes and schedules. Hence, in such contexts, path planners and navigation controllers designed for single robots are effectively applicable to multi-robot systems.

However, it is possible for the tasks to be tightly cooperative or adversarial, with dense dependencies between their trajectories. For example, in a multi-agent pathfinding problem [10, 150–153], multiple homogeneous agents move from the start to goal positions in a congested environment. Therefore, if each agent plans their path using a single-agent path planner, collisions are likely to occur. As another example, bimanual manipulation [154, 155] or multi-manipulator systems [11] typically require the cooperation of multiple agents during their tasks, such as moving a table together. Thus, even when cooperative tasks are decomposed into single-agent actions

and assigned to each arm, joint planning of their trajectories is still necessary to ensure successful collaboration.

Consequently, a task allocation framework is insufficient on its own to bridge the gap between single-agent planning and multi-agent task planning. In this chapter, we develop a learning-based intra-team trajectory planner designed for tightly cooperative tasks (as contribution 3 in Fig. 1.1). Given that object manipulation is a significant area in robotic tasks, we will demonstrate this approach using a multi-arm manipulation task as an exemplary case of tightly cooperative tasks.

The remainder of this paper is structured as follows: Sec. 5.1 motivates the research problem and outlines the key contributions of this chapter. Sec. 5.2 reviews related literature, highlighting the gaps that our study aims to address. Section 5.3 elaborates on the baseline learning framework and introduces our proposed graph neural network (GNN)-based multi-agent reinforcement learning (MARL) for tightly cooperative manipulation tasks. In Sec. 5.4, we evaluate, analyze, and compare the network structures and training setups. Finally, Sec. 5.5 concludes the chapter and discuss potential future directions.

Supplementary videos for this project can be found at:
https://brg.engin.umich.edu/publications/marl-manipulation.

5.1 Introduction

Both optimization and learning-based algorithms have been extensively used to plan trajectories for multi-agent systems. These applications span a wide range of areas, including formation control of quadcopters [156], multi-agent pathfinding [10, 150–153], multi-arm manipulation [11], target tracking [157], and multi-robot exploration [17–24].

Most existing research in this field concentrates on teams of homogeneous robots, primarily due to the increased complexity introduced by heterogeneous robot teams. Suppose there are $\mathcal{O}(N)$ types of robots, then there are $\mathcal{O}(N^2)$ pairwise cooperation types between these different robots. Since optimization-based frameworks rely on cooperation models for their planning process, the

number of models required can grow quadratically and become intractable. On the other hand, most learning-based frameworks train a joint policy for the entire multi-agent system. Such centralized frameworks are akin to single-robot learning problems. In multi-agent learning problems, each individual uses one neural network to generate their actions. Since their decisions are made separately by their own network, the structure lacks the capability to represent mutual constraints that exist in joint policies. In addition, as all agents share a single neural network, their policies remain homogeneous. Traditional MARL frameworks lack the ability to accommodate policy heterogeneity explicitly [158].

Many existing works for MARL operate under the assumption that the joint task contribution can be factorized into a product of contributions from individual agents [159, 160]. This allows for the value or reward to be factorized into a sum of individual values or rewards [161]. The resulting reward and action pairs are then utilized to train each agent's individual policy network. Such factorization effectively decouples the system, thereby accelerating the training process. However, this approach encounters limitations when applied to tightly cooperative tasks, as these tasks cannot be fully factorized. Consequently, it becomes crucial to develop a mechanism that encodes the collaboration constraints among agents' behaviors directly into their individual rewards, especially when learning tightly cooperative multi-agent tasks.

In this chapter, we introduce a novel reinforcement learning framework based on graph neural networks, designed for tightly cooperative multi-agent manipulation tasks. The communication among agents is achieved through graph convolution and can be trained during the learning process. The network structure allows for pairwise constraints between agents through communication. The design of the reward function is twofold: it includes both an individual reward, pertinent to each agent, and a global reward, which is focused on achieving joint goals. Note that our current model considers a team of homogeneous agents, with the prospect of extending this framework to include heterogeneous agents as an avenue for future research.

The contributions of this chapter are listed below.

1) The design of a reward function for each agent while encoding the constraints for a cooper-

ative manipulation task.

2) The development of a distributed GNN-based structure that efficiently manages task constraints through graph-based communications.

3) A comprehensive comparison against baseline algorithms, including factorized MARL and fully centralized joint policy learning.

5.2 Related Work

In this section, we review the related work, highlighting their strengths and weaknesses, and restate the research gaps that our study aims to address.

5.2.1 Optimization-based Methods for Tightly Cooperative Multi-agent Tasks

Most optimization-based multi-agent trajectory planners for tightly cooperative tasks are built for specific applications. Due to the complexity of the problem, a model is needed for each specific type of agent interaction.

Multi-agent pathfinding tries to find collision-free paths for multiple agents navigating in a crowded environment [10, 150–153]. This process typically involves iterative A* searches with added constraints to prevent collisions, continuing until there is no conflict among the paths of the agents. Advanced tree-style data structures are utilized to guide these iterative A* searches. For tightly coupled bimanual manipulation tasks, most existing optimization-based methods employ dual-arm kinematics and dynamics models [154, 155]. These methods combine the states and actions of two arms into one joint state space and use optimization to find a joint plan for both arms simultaneously. Similar joint state models are applied in [7, 162], where the parameters for the dynamics are learned from data. In [11], the authors innovatively integrate the concepts of the probabilistic roadmap, mode graph, and precedence-constrained tasks to represent a multi-arm

assembly task. They use mixed-integer programming and multi-agent pathfinding techniques in a hierarchical planner to create collision-free trajectories for robot arms.

Once the model is constructed, optimization-based methods can efficiently generate trajectories for tightly cooperative tasks. However, since each type of collaboration requires a specific model, it is hard for optimization-based methods to be applied to heterogeneous agents and tasks. In such settings, there can be a large number of different collaboration scenarios, which all require a collaboration model.

5.2.2 Learning-based Methods for Tightly Cooperative Multi-agent Tasks

The most straightforward approach to learning in multi-agent tasks involves concatenating the states and actions of all agents into one joint state and action vector. Then, this joint state vector is fed into a neural network to produce the corresponding action vector. This technique allows for the application of existing single-agent learning methods to train the policy network for multi-agent systems. This technique is generalizable and effective and, consequently, has been widely applied to various multi-agent tasks. The authors of [163] and [164] have successfully implemented reinforcement learning algorithms across multiple manipulation tasks, including bimanual manipulation. The authors of [144] employed similar strategies to train a joint policy for the four agents in a hide-and-seek game. However, this method has its limitations. The number of agents and the size of the state space are fixed, resulting in a rigid network structure that struggles to adapt to teams of varying sizes. Moreover, in each simulation or experimental step, only a single pair of joint state and action data is collected for training, which can lead to inefficiencies in the training process. Consequently, when dealing with large team sizes, the network's scale and the associated training overhead can become overwhelming.

In multi-agent reinforcement learning, policy networks are trained individually for each agent, diverging from using a joint policy network for the entire system. This method assigns a smaller, dedicated policy network to each agent, responsible for generating its own plan. Therefore, in each

step, N[1] pairs of state and action data can be collected to train each agent once. When the agents are homogeneous, they share one policy network. This sharing ensures that the number of network parameters remains constant, regardless of team size variations. Consequently, this approach offers a scalable and flexible solution, adaptable to teams comprising different numbers of agents, while maintaining efficiency in both network size and training methodology.

Assuming that each agent independently contributes to the overall joint reward, this reward can be factorized into a sum of individual rewards. These individual rewards are then utilized to train their respective policy networks. In [156], the researchers employed an individual reward comprising take-off, hover, and leader-follower rewards. This reward structure was instrumental in training a multi-agent formation controller specifically designed for teams of quadcopters.

Complete factorization in multi-agent reinforcement learning is feasible only when the task can be decoupled such that each agent's task is independent of the states and actions of other agents. This assumption often fails in the context of tightly cooperative tasks. An individual policy network, which bases its action solely on the state of the current agent, may struggle to effectively cooperate with others due to its inherent limitations in considering the broader team dynamics. Graph neural networks (GNNs) offer a solution to this challenge. They maintain the scalability of team sizes by allowing different agents to share the same policy network. More importantly, GNNs facilitate inter-agent communication, an essential component in cooperative scenarios. By leveraging a GNN-based approach in factorized multi-agent reinforcement learning, it is possible to enhance both the efficiency of training and the flexibility of network structure, while also supporting effective communication within a distributed agent framework.

The works of [38, 157] have effectively supervised learning techniques to train graph neural networks for multi-agent target tracking and pathfinding tasks, respectively. Our current work aims to implement factorized MRAL using a GNN framework. This approach is tailored to address the challenges of multi-agent manipulation, leveraging the strengths of GNNs in handling dynamic and interactive agent environments.

[1] N is the number of agents.

5.3 Factorized Learning for GNN-based MARL

In this section, we will use the scenario of multi-agent table lifting as a practical example to demonstrate the application of our framework. Our discussion begins by exploring learning within a joint action space. Following this, we delve into how conventional multi-agent reinforcement learning (MARL) techniques decompose the reward structure. Finally, synthesizing insights from both joint space learning and reward factorization, we introduce a Graph Neural Network (GNN)-based MARL framework. This proposed framework facilitates scalable and effective MARL, particularly for tasks requiring close cooperation among agents.

5.3.1 Problem Description

This chapter is dedicated to implementing tightly cooperative manipulation tasks through multi-agent reinforcement learning. While the primary focus is on manipulation, it is important to note that the framework proposed in this work is generalizable. It can be effectively adapted to a variety of other tightly cooperative tasks, extending its applicability beyond the specific context of manipulation.

Suppose there are multiple robotic agents and denote the agent indices as the set $I = \{1, \cdots, N_I\}$. Suppose the state of the object to manipulate is denoted as $o \in \mathcal{O}$, including object position, orientation, speed, and other possible observations. Let the state of each agent be $s_i \in \mathcal{S}_i, \forall i \in I$, including the joint angles, end effector position, velocity orientation, and other possible agent states. Let the action that an agent can perform be $a_i \in \mathcal{A}_i$, which typically contains joint angle velocities or joint angle positions. The goal of the manipulation task is to move the object from a start state o^0 to a desired goal state o^G. Let the object state, agent state, and agent action at time t be o^t, s_i^t, and a_i^t, respectively. Then, a joint policy, parameterized by weights θ, can be denoted as $\pi_\theta(o^t, s_1^t, \cdots, s_{N_I}^t) = [a_1^t, \cdots, a_{N_I}^t] : \mathcal{O} \times \mathcal{S}_1 \times \mathcal{S}_{N_I} \rightarrow \mathcal{A}_1 \times \mathcal{A}_{N_I}$. The learning algorithm needs to find a joint policy π for the multi-agent system such that the plans given by the policy, $(a_i^0, a_i^1, \cdots, a_i^{T-1}), \forall k \in K$, would move the object to its goal state within the planning

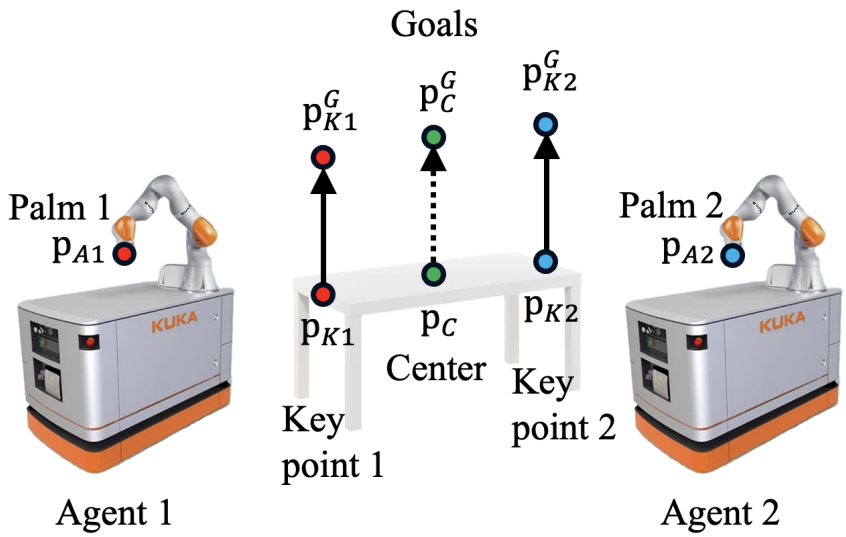


Figure 5.1: Table lifting task.

horizon, $t = 0, \cdots, T$, while maximizing the total reward gained by the agents. Note that the joint reward, $r(o, s_1, \cdots, s_{N_I}, a_1, \cdots, a_{N_I}) : \mathcal{O} \times \mathcal{S}_1 \times \mathcal{S}_{N_I} \times \mathcal{A}_1 \times \mathcal{A}_{N_I} \to \mathbb{R}$, is a scalar function of the states and actions, describing whether an action efficiently achieves the goal.

One example task is shown in Fig. 5.1. In this table-lifting task, multiple manipulators need to collaborate, lift the table, and move the table to multiple goal positions in a goal trajectory. During the task execution, the manipulators have to keep the balance of the table. This multi-robot task is tightly cooperative since the trajectory of one manipulator depends on and affects the trajectories of other manipulators during the lifting, moving, and balancing.

5.3.2 Global Reward and Joint Policy Learning

Given a fixed number of agents, such a multi-agent task can be formulated as a single-agent task where a joint policy, $\pi_\theta(o^t, s_1^t, \cdots, s_{N_I}^t)$, can be learned to control the robot.

One global reward, $r(o, s_1, \cdots, s_{N_I}, a_1, \cdots, a_{N_I}) = r_g(o, s_1, \cdots, s_{N_I}, a_1, \cdots, a_{N_I}) : \mathcal{O} \times \mathcal{S}_1 \times \mathcal{S}_{N_I} \times \mathcal{A}_1 \times \mathcal{A}_{N_I} \to \mathbb{R}$, is used to guide one giant neural network whose input contains the states of all agents $o, s_1, \cdots, s_{N_I}$ and outputs the actions for all agents $a_1, \cdots, a_{N_I}$. This is illustrated in Fig. 5.2a

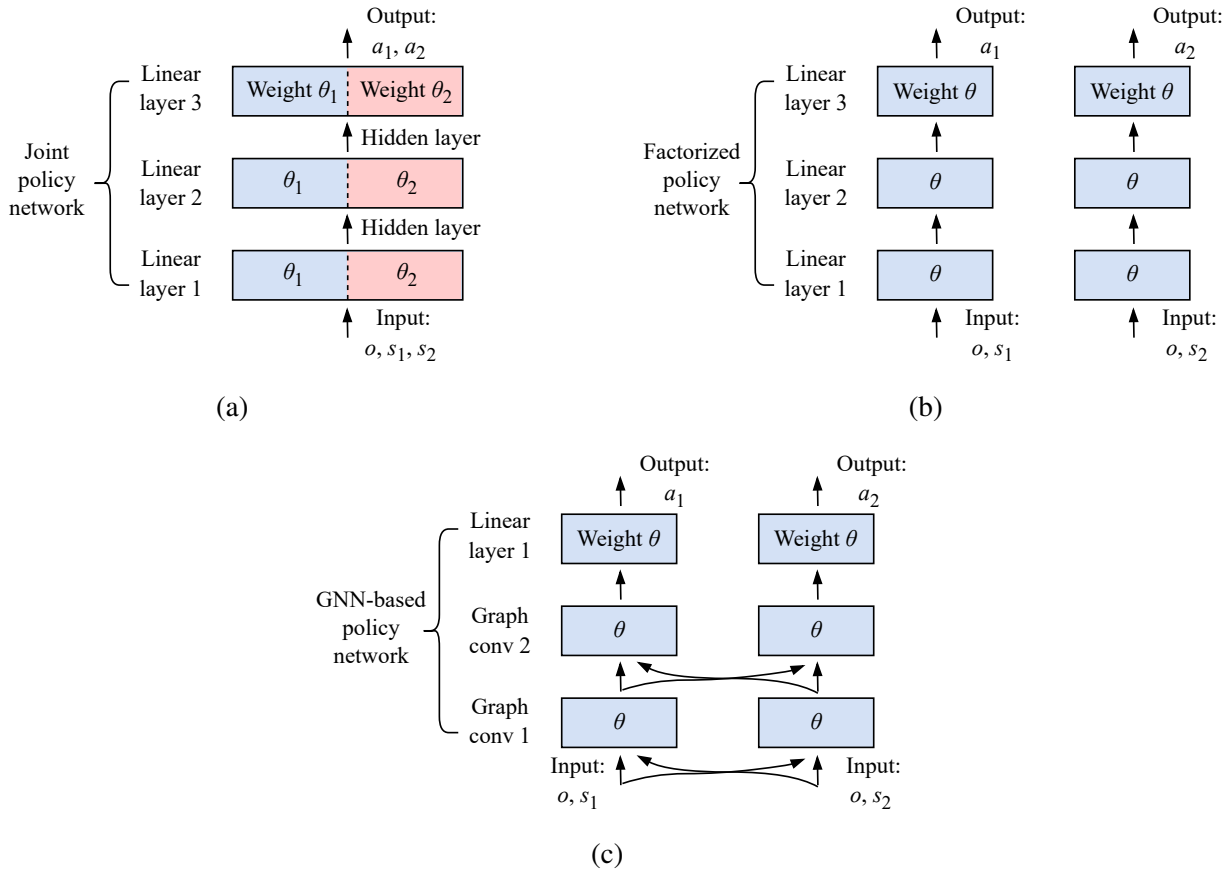


Figure 5.2: A comparison of the policy network structures and the rewards: (a) joint policy learning, (b) factorized learning, (c) GNN-based learning.

The population-based training mechanism (PBT) with a proximal policy optimization (PPO) algorithm in [164] can be used to train such a joint policy. To consider the environmental and system uncertainties, the stochastic policy π_θ is modeled as a function that maps the states to the distribution of optimal actions. In Fig. 5.3, the output distribution is modeled as a Gaussian distribution parameterized by its mean μ_i and standard deviation $\sigma_i, \forall i \in I$.

First, a recurrent neural network (RNN) summarizes the inputs at different time frames into a vector. Then, a multi-layer perception (MLP) further processes the intermediate summarized information into a condensed understanding. Finally, three separate one-layer neural networks generate the mean and standard deviation of the planned actions and predict the value of the current

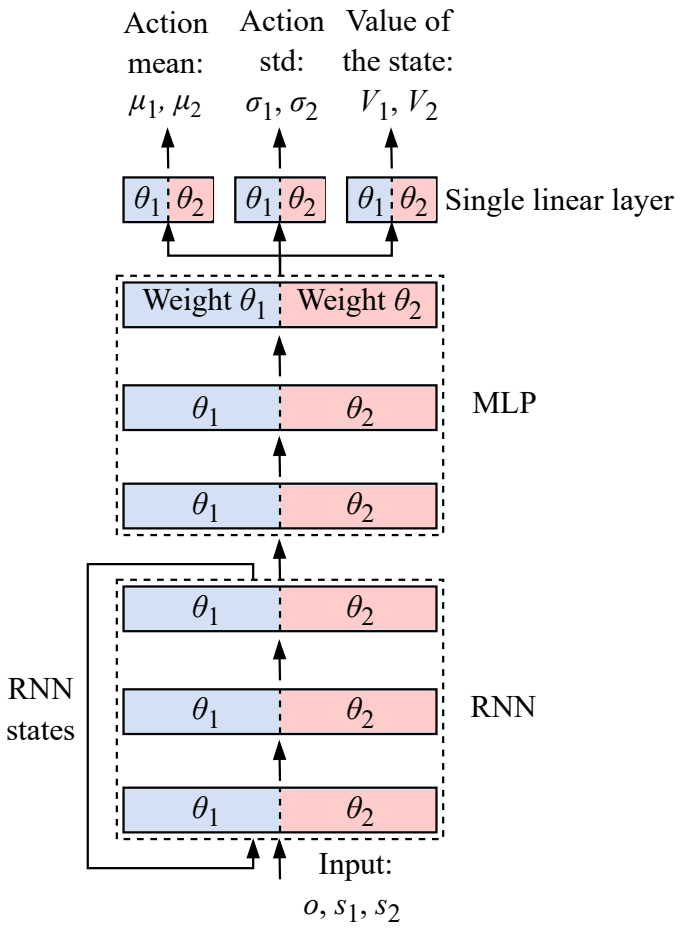


Figure 5.3: Network structure for joint policy learning.

input state. Note that the value of a state if the expected cumulative reward as calculated in (5.1).

$$V_{\pi_\theta}(o^0, s_1^0, \cdots, s_{N_I}^0) = E_{\pi_\theta}\left[\sum_0^T r_g(o, s_1, \cdots, s_{N_I}, a_1, \cdots, a_{N_I})\right] \tag{5.1}$$

Suppose π_θ is the current policy and $\pi_{\tilde{\theta}}$ is the policy after one step of update. The PPO algorithm used to update the policy population tries to minimize a clipped version of the change of the expected cumulative reward from the initial state, $E_{s_0}\left[V_{\pi_{\tilde{\theta}}}(o^0, s_1^0, \cdots, s_{N_I}^0) - V_{\pi_\theta}(o^0, s_1^0, \cdots, s_{N_I}^0)\right]$.

5.3.2.1 Joint Policy Learning for the Table Lifting Task

During the table-lifting task, the inputs and outputs for the joint policy learning are listed in Table 5.1, where N_F is the number of fingers in the end effector and N_K is the number of key

points on the objects. Note that key points are a set of points on the object used to specify the object poses during motion planning. In the current example, there are $N_F = 2$ fingers on each end effector, and key points are where the manipulators should grasp or hold the object, and therefore, $N_K = N_I$.

Table 5.1: Table-lifting task: input states and output actions for joint policy learning setup.

Category	Item	Dimension
Input: object states	Object quaternion	4
	Object velocity	3
	Object angular velocity	3
	Key point poses relative to goal	6
	Object scales in x, y, z directions	3
	Max key point distances to goal	1
	Number of lifted object (binary)	1
	Task progress	1
	Task success (binary)	1
	Reward of this step	1
Input: agent states	Arm joint angles	$9N_I$
	Arm joint angles velocities	$9N_I$
	Palm position	$3N_I$
	Palm quaternion	$4N_I$
	Palm velocity	$3N_I$
	Palm angular velocity	$3N_I$
	Finger poses relative to palm	$3N_FN_I$
	Key point poses relative to palm	$3N_I$
Output: agent actions	Target arm joint angles	$9N_I$

The reward $r_g(\cdot)$ consists of three components: palm approaching reward $r_A(\cdot)$, goal reaching reward $r_C(\cdot)$, and balancing reward $r_B(\cdot)$. Let $s^t = (s_1^t, \cdots, s_{N_I^t})$ be the union of all agent states, and $a^t = (a_1^t, \cdots, a_{N_I^t})$ be the union of all agent actions. Then, the global reward is computed according to (5.2), where c_A, c_C, and c_B are the weight for each component. The approaching reward in (5.3) is positive when the least close palm gets closer to the corresponding key point during the task execution. As illustrated in Fig. 5.1, $\mathbf{p}_{Ai}^t$ is the position of palm $i \in I$ at time t and $\mathbf{p}_{Ki}^t$ is the position of key point $i \in \{1, \cdots, N_K\}$ at time t. The goal-reaching reward in (5.4) is positive when the center of the table, $\mathbf{p}_C^t$, moves toward the goal position $\mathbf{p}_C^G$, at time t.

Suppose the center of the four legs of the table are points $\mathbf{p}_{Lj} = [x_{Lj}, y_{Lj}, z_{Lj}]^\mathsf{T}, j \in \{1, 2, 3, 4\}$. The balancing reward in (5.5) penalizes the difference between the height of the legs.

$$r_g(o^t, a^t, s^t) = c_A r_A(o^t, a^t, s^t) + c_C r_C(o^t, a^t, s^t) + c_B r_B(o^t, a^t, s^t) \tag{5.2}$$

$$r_A(o^t, a^t, s^t) = \max_i ||\mathbf{p}_{Ai}^{t-1} - \mathbf{p}_{Ki}^{t-1}|| - \max_i ||\mathbf{p}_{Ai}^{t} - \mathbf{p}_{Ki}^{t}|| \tag{5.3}$$

$$r_C(o^t, a^t, s^t) = ||\mathbf{p}_C^G - \mathbf{p}_C^{t-1}|| - ||\mathbf{p}_C^G - \mathbf{p}_C^{t}|| \tag{5.4}$$

$$r_B(o^t, a^t, s^t) = -(\max_j z_{Lj} - \min_j z_{Lj}) \tag{5.5}$$

5.3.3 Composite Individual Reward and Factorized Learning

The joint policy learning setup in Fig. 5.2a can be used to learn a centralized planner for the multi-agent system. Since a new trunk of weight θ_i is added to the policy network for each agent, the framework can easily handle heterogeneous policies among the agents. However, the framework can be too rigid as a new policy network is required if the number of agents changes. In addition, in each simulation or experiment step, only one data triple of global observation, action, and reward is collected to train the whole network once. This setup can be inefficient for multi-agent systems where agents perform similar actions.

Factorized learning is frequently applied in MARL to improve the scalability and efficiency of the framework [159–161]. As shown in in Fig. 5.2b, it assumes the reward for the multi-agent system, $r(o, s_1, \cdots, s_{N_I}, a_1, \cdots, a_{N_I})$, to be the sum of individual rewards $r_i(o, s_i, a_i)$.

$$r(o, s_1, \cdots, s_{N_I}, a_1, \cdots, a_{N_I}) = \sum_{i \in I} r_i(o, s_i, a_i) \tag{5.6}$$

In addition, when the agent shares a homogeneous policy, all agents share one policy network, $\pi_\theta(o, s_i)$, and rely on it to generate the individual plans. To be more explicit, there is only one set of policy weights, θ, for all the agents. Therefore, as illustrated in Fig. 5.2b, N_I data triples can be gathered in each simulation or experiment step to train the network that is around $1/N_I$ of the size of the joint network in Fig. 5.2a. This significantly improves the efficiency of the training process.

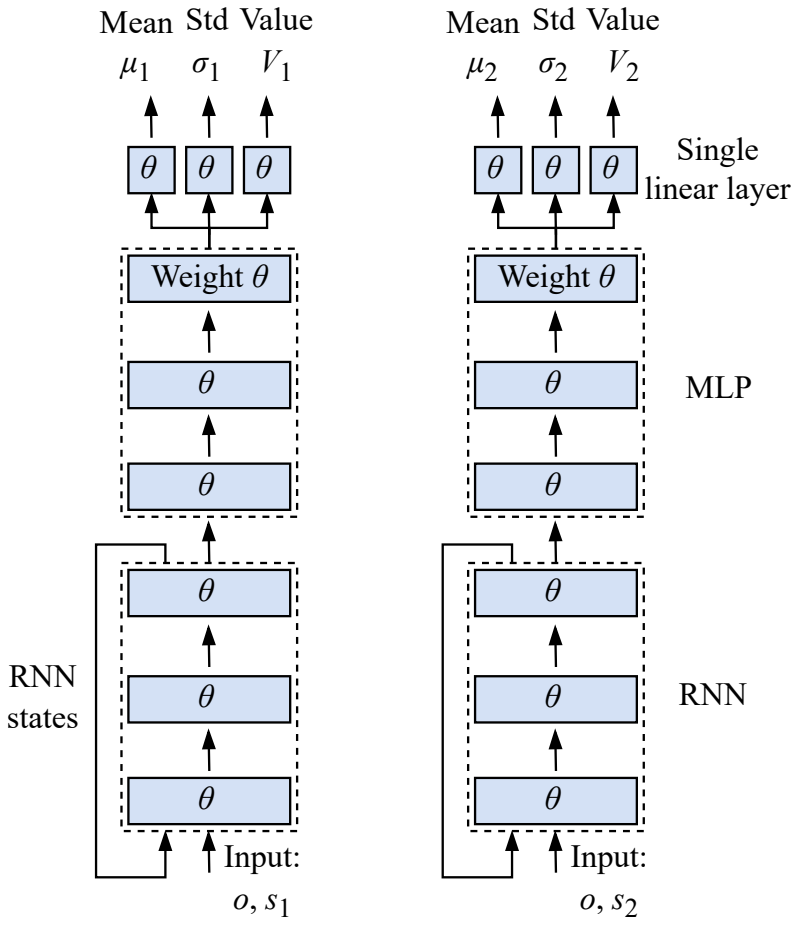


Figure 5.4: Network structure for factorized learning.

The network structure under the factorized learning setup (Fig. 5.4) is very similar to the joint learning setup (Fig. 5.3), the input is passed through an RNN, an MLP, and three separate one-layer neural network heads to generate the planned actions and the predicted values. However, the width of the network is around $1/N_I$ of the network size under the joint learning setup because the input and output are the states and actions of one individual agent.

The training is done using individual rewards and the data triple from all agents are used to train one network. Therefore, in each update step, the algorithm minimizes the clipped expected cumulative reward from the initial state, $\sum_{i \in I} E_{s_0} \left[V_{\pi_{\tilde{\theta}}}(o^0, s_i^0) - V_{\pi_\theta}(o^0, s_i^0) \right]$, where $V_{\pi_\theta}(o^0, s_1^0, \cdots, s_{N_I}^0)$

is calculated according to (5.7).

$$V_{\pi_\theta}(o^0, s_1^0, \cdots, s_{N_I}^0) = E_{\pi_\theta}\left[\sum_0^T r_i(o, s_i, a_i)\right] \tag{5.7}$$

Note that the value of one individual agent might be affected by other agents' states and actions. The factorized learning ignores those dependencies and, therefore, could result in worse task performances.

5.3.3.1 Composite Reward and Factorized Learning for the Table Lifting Task

In the table lifting task, the agents can share their policy weights because the individual agent tasks are homogeneous. Consider the example table lifting task in Fig. 5.1. The poses of the two agents are symmetrical.

The inputs and outputs for the factorized policy learning are listed in Table 5.2. Recall that N_K is the number of key points on the objects, and it is assumed that it equals the agent number, $N_K = N_I$. In factorized learning, it is assumed that each key point corresponds to one agent, and that is where the agent should grasp. Therefore, the key point-related states are categorized as agent states instead of object states. Note that the policy is homogeneous only if the input in Table 5.2 are in the agents' local coordinate systems.

The reward $r_i(o^t, a_i^t, s_i^t)$ for agent i consists of three components: palm approaching reward $r_A(a_i^t, s_i^t)$, goal reaching reward $r_C(a_i^t, s_i^t)$, and balancing reward $r_B(o^t)$, and is computed according to (5.8). Since the reward $r_A(a_i^t, s_i^t)$ and $r_C(a_i^t, s_i^t)$ only depend on states of the current agent, it is an individual reward component. $r_B(o^t)$ is a function of the global states, and it is a global reward component. Therefore, this is a composite reward as it contains both individual and global rewards for each agent.

Table 5.2: Table-lifting task: inputs and outputs for factorized and GNN-based learning.

Category	Item	Dimension
Input: object states	Object quaternion	4
	Object velocity	3
	Object angular velocity	3
	Object scales in x, y, z directions	3
	Number of lifted object (binary)	1
	Task progress	1
	Task success (binary)	1
	Reward of this step	1
Input: agent states	Arm joint angles	9
	Arm joint angles velocities	9
	Palm position	3
	Palm quaternion	4
	Palm velocity	3
	Palm angular velocity	3
	Finger poses relative to palm	6
	Key point poses relative to palm	3
	Key point poses relative to goal	3
	Key point distances to goal	1
Output: agent actions	Target arm joint angles	9

$$r_i(o^t, a_i^t, s_i^t) = c_A r_{Ai}(o^t, a_i^t, s_i^t) + c_C r_{Ci}(o^t, a_i^t, s_i^t) + c_B r_{Bi}(o^t, a_i^t, s_i^t) \tag{5.8}$$

$$r_A(a_i^t, s_i^t) = ||\mathbf{p}_{Ai}^{t-1} - \mathbf{p}_{Ki}^{t-1}|| - ||\mathbf{p}_{Ai}^{t} - \mathbf{p}_{Ki}^{t}|| \tag{5.9}$$

$$r_C(a_i^t, s_i^t) = ||\mathbf{p}_{Ki}^{G} - \mathbf{p}_{Ki}^{t-1}|| - ||\mathbf{p}_{Ki}^{G} - \mathbf{p}_{Ki}^{t}|| \tag{5.10}$$

$$r_B(o^t) = -(\max_j z_{Lj} - \min_j z_{Lj}) \tag{5.11}$$

The approaching reward in (5.9) is positive when the palm of the agent gets closer to the corresponding key point. $\mathbf{p}_{Ai}^t$ is the position of palm $i \in I$ at time t and $\mathbf{p}_{Ki}^t$ is the position of key point $i \in \{1, \cdots, N_K\}$ at time t. The goal-reaching reward in (5.10) is positive when the key point, $\mathbf{p}_{Ki}^t$, moves toward the goal position $\mathbf{p}_{Ki}^G$. The goal-reaching reward is no longer based on the center of mass in (5.4). However, if all the key points on the objects move toward the goal, then the center of mass will also move to the goal. Recall the center of the four legs of the table are points $\mathbf{p}_{Lj} = [x_{Lj}, y_{Lj}, z_{Lj}]^\mathsf{T}, j \in \{1, 2, 3, 4\}$. The balancing reward in (5.11) remains the same. Balance reward is a global reward that encodes the tightly cooperative task constraints between multiple agents to maintain balance while moving the table.

5.3.4 Composite Individual Reward and GNN-based Learning

In a fully factorized learning framework, the reward of one agent, $r_i(o, s_i, a_i)$ and the policy $a_i = \pi_\theta(o, s_i)$, is assumed to be independent of the states or actions of other agents. The network structure and reward in Fig. 5.2b shows this independence. In practice, the task performance depends on the actions of all agents. Therefore, the global part of the reward can depend implicitly on the actions of other agents, and the independence assumption does not hold. When the actions of the agents depend on each other, the network structure in Fig. 5.2b and Fig. 5.4 lack the capability to capture such dependence because the action output a_i is a function of only the states o and s_i, but not a_j or s_j, $(i \neq j \in I)$.

The joint learning network in Fig. 5.2a can easily handle inter-agent task dependencies through

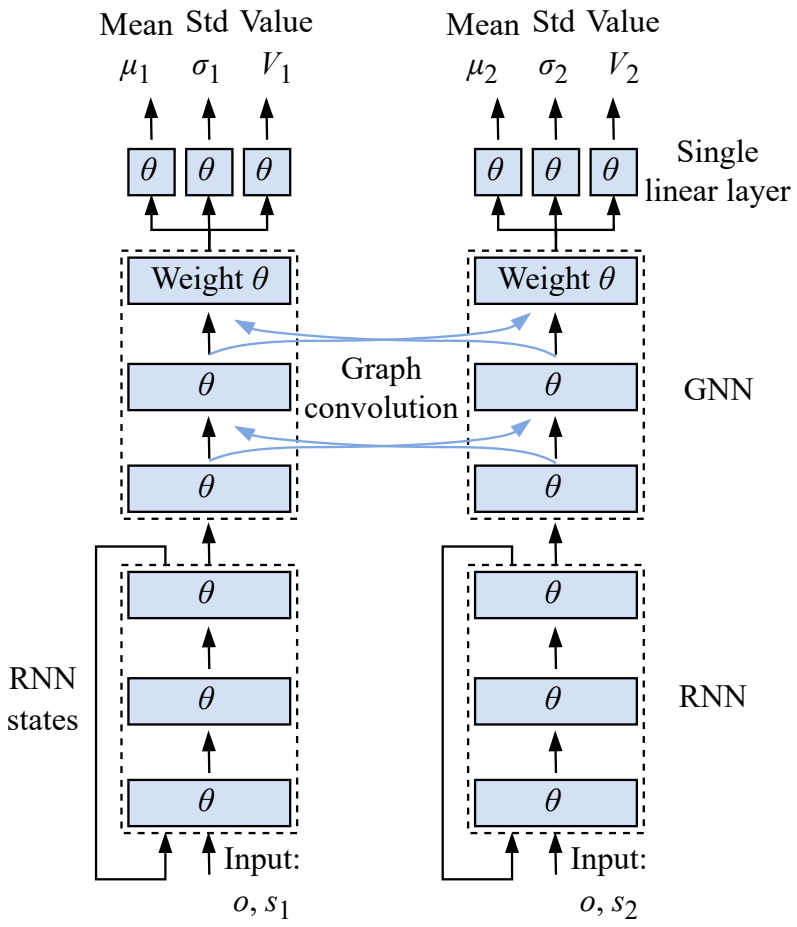


Figure 5.5: Network structure for GNN-based reinforcement learning.

a fully connected network structure. However, a fully connected network is rigid, not scalable to more agents, and can be inefficient during the training. Considering the inter-agent task dependencies and the scalability, we propose a graph neural network-based structure to learn agent communication, handle the task dependencies, and remain scalable.

The policy network structures and reward setups are compared in Fig. 5.2. The GNN-based policy network can handle the task dependencies among agents through learned communication. In the meantime, it can also be trained efficiently using individual rewards.

The network structure for GNN-based learning is illustrated in Fig. 5.5. A GNN replaces the multi-layer perception in the factorized learning in Fig. 5.4. The graph convolutional layers enable communication between agents. Therefore, the output actions, a_i, of agent $i \in I$ are a function of

the object states o, agent states s_i, but also the states of other agents s_j ($j \in I$ and $j \neq i$).

5.3.4.1 GNN-based Learning for the Table Lifting Task

The inputs and outputs for the table lifting task are the same as factorized learning and listed in Table 5.2. In addition, the composite individual reward functions (5.8)-(5.11), developed in Sec. 5.3.3.1, are applied to train the GNN-based policy network efficiently.

5.3.5 Comparison Between Network Structures

A comparison of the policy network structures and the reward functions of the three training setups have been illustrated in Fig. 5.2. The advantages and disadvantages of the three setups are summarized in Table 5.3. The JNT model is able to handle heterogeneous agent policies and dense agent communications. However, since the network structure depends on the number of agents, N_I, once trained, it can only be applied to a multi-agent system with a fixed number of agents. In addition, it uses one reward to train the network for the whole multi-agent system and, therefore, can be inefficient during the training. The FCT model requires a much smaller number of parameters, is flexible in the team size, and uses composite individual rewards to train the network for each agent efficiently. However, there is no communication among the agents. The GNN model combines the advantages of the JNT and FCT model. It automatically learns distributed agent communication through graph convolution and shares hidden layers that summarize knowledge from the agents.

Table 5.3: A comparison of the network structures. N_I is the number of agents. $N_{\mathcal{S}}$ and $N_{\mathcal{A}}$ are the number of states and actions of one agent, respectively, which are also the input and output of the policy network.

Structure	Network parameters	Agent number	Reward number	Agent Communication
JNT	$\mathcal{O}(N_I N_{\mathcal{A}} N_{\mathcal{S}})$	Fixed	One	Centralized
FCT	$\mathcal{O}(N_{\mathcal{A}} N_{\mathcal{S}})$	Flexible	Multiple	No
GNN	$\mathcal{O}(N_{\mathcal{A}} N_{\mathcal{S}})$	Flexible	Multiple	Distributed

5.4 Simulation Results

In this section, we use the table lifting task (Fig. 5.1) to evaluate the training progress and task performance of the three different network structures and reward setups in Sec. 5.3.

Our experiments are conducted using instances with 8 CPU cores and a single laptop Nvidia 3060 GPU with 16 Gb of VRAM. 2048 parallel environments are simulated using the Isaac Gym engine [165]. With the GPU-based vectorized RL implementation (rl_games[2]), 3.5×10^4 training samples (or environmental steps) can be gathered per second. According to the experiments, training a successful policy takes around 1×10^9 environmental steps and around 8 hours using the population-based training algorithm in [164].

The population-based training algorithm in [164] leverages the idea of evolutionary algorithms. It simultaneously trains N_P policies using the PPO algorithm. After a fixed number, N_{iter}, of environmental steps, the fitness scores of the policies are evaluated, and mutation and crossover operations are performed to the weights of the policies to reproduce the next generation (also called an iteration). In this experiment, $N_P = 8$ and $N_{\text{iter}} = 2 \times 10^7$. Recall that the number of environmental steps needed to train a successful policy is around 1×10^9, which means around 50 iterations are needed.

For the table lifting task, the agents need to lift the table and move it to multiple goal positions in a trajectory. Therefore, the cumulative reward gathered in each task execution, the number of lifted tables, and the number of goals reached during the task are used as metrics to evaluate the task performance. Fig. 5.6-5.8 shows the task performance along the training process.

Consider the number of iterations needed for the curves to reach their peak as a metric of how fast the policy network is trained. In Fig. 5.6-5.8, we see a rank of training speeds as follows: FCT > GNN > JNT. Note that the speed difference is not large because the case in the figures involves two robots. In future work, we will increase the agent number to provide more evidence. Factorizing one joint reward into multiple individual rewards can improve the efficiency of policy learning in the table-lifting task. In addition, more complex (more layers or more parameters)

[2] https://github.com/Denys88/rl_games

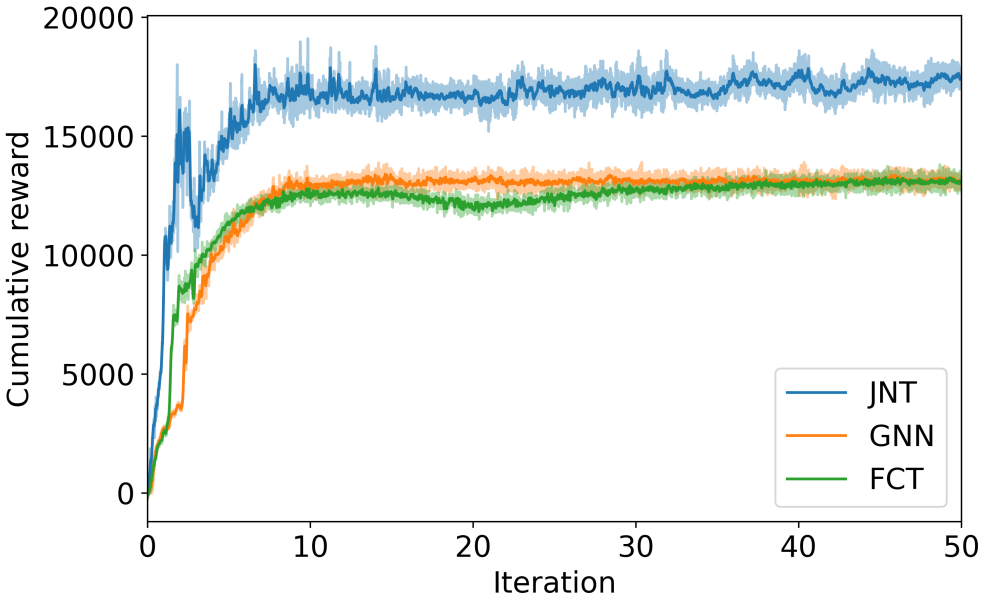


Figure 5.6: Cumulative reward for the table lifting task.

network structures lead to lower training speeds.

As shown in Fig. 5.7, all three network structures can be trained to lift almost all the 2048 tables within 2-3 iterations. According to the reached goals per environmental step in Fig. 5.8, JNT is able to achieve more goals within the unit time, which indicates it learns a more effective policy for the task. The success of the JNT model may be due to the larger number of parameters in its network and the capability to learn heterogeneous policies for the agents. The GNN and FCT models achieve fewer goals within the unit time, with GNN achieving more goals than FCT. The better performance of GNN is possibly due to its capability to enable agent communication.

Though the GNN and FCT models achieve fewer goals, as mentioned in Table 5.3, they can be applied to multi-agent teams with different numbers of agents[3], archive a distributed framework, and can be more efficiently trained because multiple individual rewards are used to train a smaller-sized network.

We demonstrate the multi-agent table lifting task in supplementary videos[4].

[3]This statement is currently grounded in theory, based on the inherent properties of GNN. The simulation experiment in this study evaluated teams comprising only two agents. Future studies will vary team sizes to further substantiate this claim.

[4]https://brg.engin.umich.edu/publications/marl-manipulation

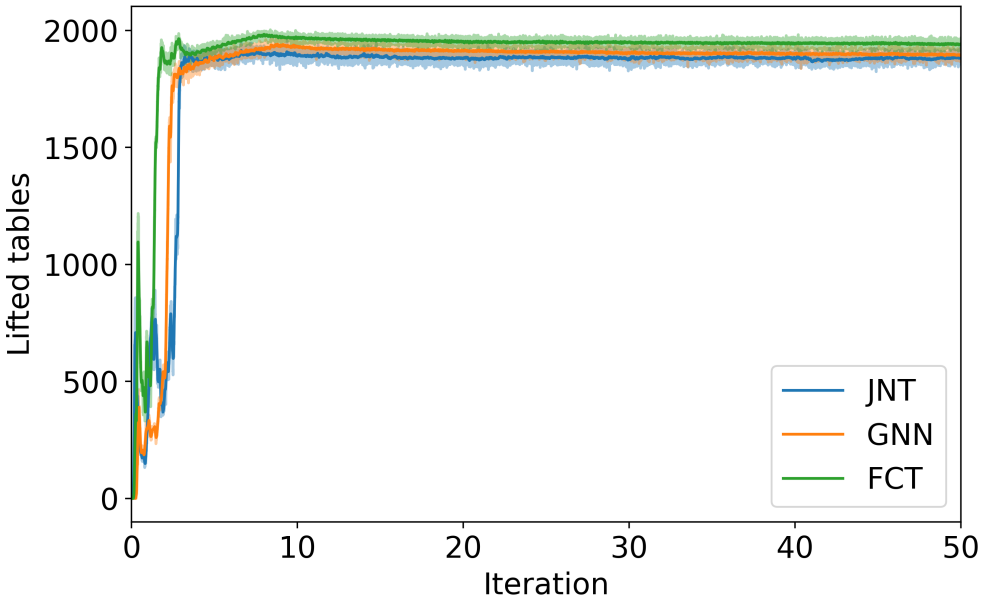


Figure 5.7: Lifted tables in each environmental step for the table lifting task.

5.5 Conclusions and Future Work

5.5.1 Conclusions

This chapter introduces a novel multi-agent reinforcement learning (MARL) framework designed for executing tightly cooperative tasks. Central to this framework is the concept of factorized learning, which enhances the efficiency of training policy networks within a homogeneous team of agents. A key innovation is the integration of a graph neural network (GNN) as the core of the policy network. This GNN structure autonomously learns and facilitates communication among agents, a critical component for teamwork.

The effectiveness of this framework is demonstrated through a multi-agent table-lifting task. Comparative results reveal that the proposed MARL models, specifically the GNN-based model (GNN) and the factorized model (FCT), achieve successful task completion akin to a joint policy network (the baseline, JNT model). Notably, the GNN model outperforms the FCT in terms of the number of goals reached. Furthermore, the MARL framework exhibits superior efficiency in training policy networks compared to the joint training baseline.

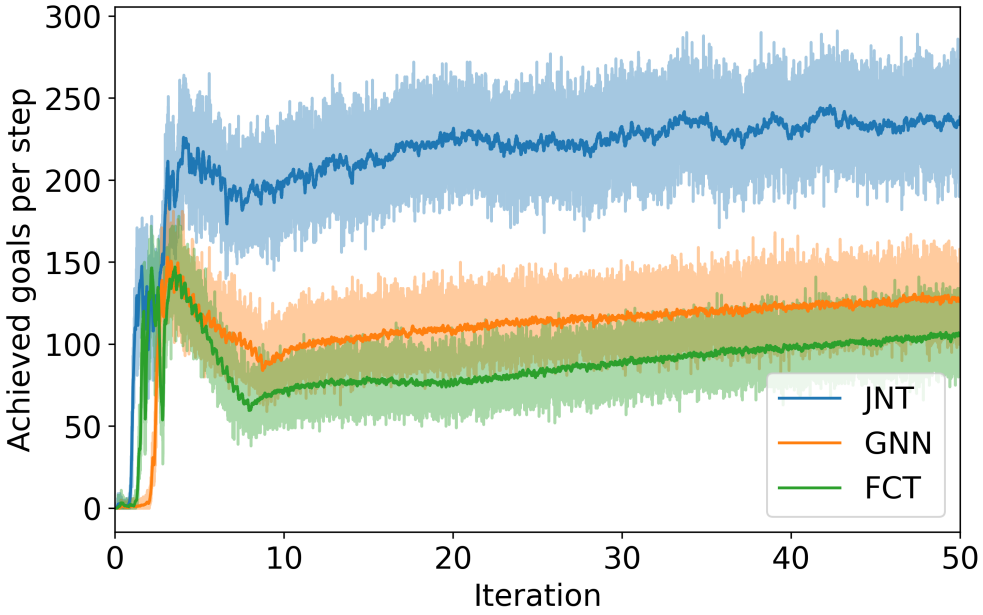


Figure 5.8: Achived goals in each environmental step for the table lifting task.

In summary, the innovative GNN-based policy network introduced in this framework supports adaptable team sizes and fosters effective communication among agents. This leads to enhanced cooperation and combines the strengths of both joint and fully factorized multi-agent reinforcement learning approaches.

5.5.2 Future Work

The GNN-based framework features a shared policy among agents, enabling flexibility in team size. Once the framework is trained with varying numbers of agents, it can adapt to different team configurations. However, further parametric studies are required to evaluate the maximum scalable team size within this framework.

When multiple robots or robotic manipulators collaborate, they can form closed kinematic chains while jointly moving an object. Consider a scenario where two manipulators are simultaneously lifting a table: as they both grasp the table, the manipulators and the table form a closed kinematic chain. This formation reduces the degree of freedom in the system, confining the movement of the manipulators to a specific manifold within the original state space. This closed kinematic

chain is not accounted for in our current system model. Consequently, the trained policy network avoids such configurations, leading the manipulators to support the table from underneath rather than securely grasping it. Incorporating closed kinematic chain considerations into the structure of a Graph Neural Network (GNN) presents a significant challenge, given the variable number of agents that can form a chain. Future research could explore ways to encode these chain constraints within GNNs, potentially enhancing training efficiency in such scenarios.

The result section evaluates the framework solely through a table-lifting task. To thoroughly assess its generalizability, it is essential to implement and test the framework across a broader range of tightly cooperative tasks.

At present, the framework operates under the assumption of agent homogeneity. An intriguing direction for future research would be to explore the potential benefits of incorporating heterogeneous agents within the team. This could lead to a more flexible framework capable of tackling diverse cooperative challenges.

CHAPTER 6

Conclusion and Future Directions

With recent advancements in sensing and control, robots have become capable of an ever-growing range of single-agent tasks. On the other hand, there is still a lack of coordination frameworks that leverage the full capability of the single agents during complex multi-agent tasks. This work tries to bridge the gap between single and multi-agent tasks through the development of a hierarchical multi-agent feedback planning system, as depicted in Fig. 1.1. A stochastic optimization-based task allocation planner is developed to simultaneously optimize task decomposition, assignment, and scheduling. It runs both in pre-execution and real-time and applies both proactive and reactive methods to generate robust plans under time, energy, and capability uncertainty. The framework scales to hundreds of agents and generalizes to multiple problem types. A learning model is formulated and serves as the feedback step in the framework to estimate the energy cost, agent capabilities, and task requirements for the task allocation planner. Considering the situations where agent actions are still not decoupled after the task allocation, we propose a reinforcement learning-based intra-team trajectory planner to generate cooperative trajectories for the heterogeneous robots. Simulation and experimental platforms are being developed, and task cases, including capture-the-flag game, robotic service allocation, multi-robot tour guiding, exploring and block picking, and cooperative manipulation are being used to evaluate the proposed framework.

In the rest of this chapter, we first recapture and summarize the contributions of this dissertation, then highlight its broader impact, and finally point out some limitations that drive potential future

extensions.

6.1 Contributions

6.1.1 Multi-agent Task Allocation for Large-scale Complex Tasks

The stochastic programming-based task allocation planner developed in this dissertation can be generalized and applied to heterogeneous practical problems.

The simultaneous optimization of the task decomposition, task assignment, task schedule, and agent routes enables the systems to generate optimal solutions for complex task allocation scenarios where all these problems are coupled.

Different uncertainties have been captured in the framework, and the related risks are minimized to provide robust teaming plans and schedules[1].

With the hierarchical structure of the planner, the choices of abstraction in each planning level, and the problem-specific algorithms to speed up the optimization[2], the developed task allocation planners can be applied to problems involving hundreds of agents and tasks, offering a significant advancement over existing frameworks.

6.1.2 Task Requirement and Agent Capability Learning

Existing optimization-based task allocation frameworks typically rely on a pre-defined, known model to describe tasks. In practice, task-related modeling parameters are hard to determine. To address this limitation, we develop a learning model to estimate the capability-based task requirements for our task allocation model. As capability-based task constraints can be represented as linear constraints, they have been applied in many existing task allocation frameworks. Therefore,

[1]The uncertainties in task requirements and agent capabilities are considered in Chapter 2 to generate robust task assignment. The uncertainty in the time costs is considered in Chapter 3 to generate robust schedules. The uncertainty in the energy cost is modeled in our work [44] to plan for robust routes.

[2]For example, the flow decomposition and L-shaped method in Chapter 2 and the large neighborhood search algorithm in Chapter 3.

the proposed learning model also improves the applicability of many existing works, enhancing the utility of our model.

The learning model we propose is grounded in a linear program, making it highly efficient: a training set comprising ten thousand samples can be processed in several seconds to estimate model parameters. This rapid processing capability facilitates real-time, closed-loop model updates for replanning purposes.

The framework is designed to be generalizable across various real-world scenarios since it only requires team configuration and task performance pairs to estimate the task requirements. This generic, problem-agnostic information can be easily obtained through simulations, ensuring the generalizability of the framework across different real-world contexts.

6.1.3 Intra-team Trajectory Planners for Tightly Cooperative Tasks

In Chapter 2, we introduce a task allocation framework that efficiently generates schedules and individual task plans for each robot agent. However, in scenarios where tasks require tight cooperation, the trajectories of the agents can intersect or interfere with one another. This complexity necessitates an intra-team trajectory planner, enabling the robots to collaborate at a trajectory level and avoid conflicts.

The diverse range of potential collaborations between robots makes it impractical to develop a specific model for each interaction type. To overcome this, we introduce a graph neural network-based multi-agent reinforcement learning framework. This innovative approach is designed to efficiently train a policy network, enable communication between agents, and enhance the scalability of the system.

Our framework features a composite reward function that balances individual and collective goals, especially pertinent to cooperative manipulation tasks. Furthermore, its distributed GNN structure capably handles task constraints through direct pairwise communications among agents, enhancing both efficiency and adaptability.

6.2 Broader Impact

6.2.1 Comprehensive Hierarchical Multi-agent Framework

This work tries to bridge the gap between single-agent and multi-agent systems by answering the following research questions:

1) What additional components are necessary to transition from single-agent to multi-agent task planning and control systems?

2) What constitutes the optimal hierarchical framework?

3) At what level of abstraction should each component within this framework operate for optimal task performance?

Addressing these questions, the study acknowledges the significant computational demands of integrating all components of a multi-agent system—from perception to control—within a singular optimization process. As a result, the adoption of a hierarchical planning framework is not just beneficial but essential.

This dissertation presents an effective hierarchical framework that groups and optimizes the various sub-problems inherent in multi-agent systems, including task assignment, scheduling, joint trajectory planning, and task requirements learning. The framework emphasizes modularity, allowing for the seamless integration and reuse of existing single-agent planning and sensing techniques.

It enhances the knowledge of integrating the multiple levels of a multi-agent system in one consistent framework. While the immediate application of this research is in multi-robot cooperation, its implications extend far beyond. The developed framework and the insights gained have the potential to influence various fields, including emergency response, healthcare, urban planning, agriculture, financial decision-making, and smart manufacturing collaboration.

6.2.2 Heterogeneous and Generalizable Multi-agent System

While most current research on heterogeneous multi-agent tasks is tailored for specific scenarios, this work focuses on enhancing the modeling of heterogeneous agents and tasks, emphasizing generalizability.

A core concept in this framework is the usage of task requirements and agent capabilities. This model acts as task behavior constraints, making it adaptable to various multi-agent tasks. It not only addresses the complexity inherent in such tasks but also ensures generalizability across different scenarios.

This approach may pave the way for more reusable and adaptable systems that can seamlessly adjust to varying conditions and requirements. Such a shift is instrumental in driving the field towards a future where multi-agent systems are not just effective in their designed scenarios but are flexible enough to meet the challenges of a rapidly evolving landscape.

6.2.3 Robust and Adaptive Multi-agent System

This work advances the modeling of uncertainty and managing disturbances within multi-agent systems. A critical aspect is the selection of appropriate risk metrics. These metrics are pivotal in addressing the inherent uncertainties in task requirements and agent capabilities, enhancing the robustness across various scenarios.

A key concept lies in the dual strategy employed for planning. In the pre-execution stage, the focus is on robust, proactive planning. This phase emphasizes anticipating and mitigating potential uncertainties and disturbances before they occur. During execution, the approach shifts to reactive, adaptive planning. This ensures that the system can respond effectively to real-time challenges, adjusting its strategies as needed.

By integrating both proactive and reactive elements into the planning process, this methodology can foster robust and adaptive solutions to a range of scenarios. The approach outlined in this work thus significantly contributes to tackling the complexities of uncertainty and disturbance in multi-agent environments.

6.3 Future Directions

6.3.1 Distributed Multi-agent Task Allocation

The proposed framework formulates the task allocation problem as one mixed-integer optimization and solves it in a central planner. Such a centralized framework can guarantee optimality and improve the solution quality. However, it places all computational load on one central planner and can lead to scalability issues and vulnerability to point failure. In addition, it presumes the existence of reliable low-latency communication. While some of the preconditions of a centralized framework can be achieved during the pre-mission planning stage when conditions are controlled, the efficacy of the framework may be reduced once agents are dispersed across various locations for task execution. In such scenarios, the central planner's capacity to adapt to real-time feedback in the replanning stage is challenged, highlighting the need for a more flexible approach in dynamic, real-world environments. Consequently, a distributed framework may be a preferable option. It further enhances the scalability, improves the robustness against point failures, and reduces the communication overhead.

Auction-based (market-based) and consensus-based algorithms are widely used for multi-agent task allocation. In a consensus-based framework [166–168], each agent chooses the next task based on their individual planner. Then, they communicate iteratively to deconflict such that no task will be fulfilled twice. Most consensus-based algorithms are designed to converge to a conflict-free global plan with a fixed or finite number of iteration steps. In an auction-based framework [57], a task is first assigned to an auctioneer, one of the agents. This agent then collects the bids from other agents, which contain the costs those agents require to complete the particular task. When the bidding stage is finished, the auctioneer will assign the task to the agent who has the minimum cost bid (plan) for completing the task. To improve the efficiency of the assignment and handle the dependency between tasks, bundle-based auction algorithms first group multiple related tasks into bundles and then assign the bundles to agents through combinatorial auctions [169].

While bundle-based auction algorithms are suitable for single-agent tasks with dependencies,

they still assume the assignment between agents is independent. For complex multi-agent tasks defined in Chapter 1 and 2, where dependencies exist not only between tasks but also among agents, a more innovative approach is proposed. This involves grouping both agents and tasks into bundles, thereby creating a temporary centralized planner akin to an auctioneer for a limited scope of agents and tasks. This strategy allows for local modifications of the original plan, aiming to achieve an improved plan. Based on the theory of large neighborhood search, this approach will reach local optimal solutions if this process is iteratively performed. Considering the interdependencies of tasks and agents, this approach will offer a refined solution to the challenges presented in complex multi-agent task allocation scenarios.

6.3.2 Real-time Replanning for Task Allocation

There are two primary strategies to deal with uncertainty and disturbances in dynamic systems: robust proactive planning in the pre-execution stage and reactive adaptive planning during execution. Robust proactive planning anticipates and prepares for potential disturbances before they occur, while reactive adaptive planning focuses on restoring system stability after disturbances have occurred during the execution. The task allocation framework developed in Chapter 2 and 3 evaluates the potential uncertainty in the environment during the pre-execution state to generate robust plans.

With the learning framework developed in Chapter 4 serving as the feedback, it becomes feasible to develop a real-time replanning system to adjust the plan in response to updated information, particularly in scenarios of agent or task failures.

The replanning process is engineered with two key objectives. First, it aims to implement a partial update that preserves the integrity of the completed and ongoing segments of the original plan. This approach minimizes disruptions while ensuring that the additional costs incurred to recover from failures are kept low. Second, the replanning optimization should be solved efficiently, enabling it to update the plan in real-time during execution.

To achieve the partial update, the task allocation problem can be reformulated as a constrained

optimization where the completed plan is forced to be consistent with the original plan. The work of [170] develops a vehicle routing system for dynamic customer pickups. Additional constraints are added during their replanning process to ensure that the vehicles must serve all committed orders. One essential aspect is constraining the replanning scope exclusively to the agents related to the failed task to minimize the change to the original plan, thereby maintaining overall plan integrity. Another prevalent approach in task allocation is to remove the completed tasks from the optimization and focus on completing the remaining tasks from the current system state. In [171], tasks to be completed are not fully known at the beginning and can dynamically appear. When a new task appears during execution, their sequential single-item auction method tries to insert the task into the plan with the lowest additional cost based on the current robot locations.

In practice, a low-cost recovery plan might not exist if the possibility of failure is ignored and a backup plan is not evaluated in the pre-execution stage. Therefore, a replanning penalty could be added to the pre-execution planner to favor the plans with low recovery costs. Many stochastic vehicle routing systems adopt this strategy by defining recourse actions to take when failure happens and minimizing the expected recourse cost during pre-execution [120, 172–175]. For instance, the work [120] deals with stochastic demand in a routing problem. When a vehicle has no additional resources to satisfy demand at a location, it returns to the depot to reload before servicing the next customer. The expected reload cost due to failures is minimized pre-execution. Likewise, [175] employs a methodology where the expected additional traffic flow changes due to uncertainties in demand are penalized. This penalization helps select an initial plan that requires minimal adjustments in the face of disturbances. These approaches underscore the significance of anticipating and planning for uncertainties in dynamic operational environments.

Many multi-agent task allocation problems [29, 47–51] are NP-hard and time-consuming to solve even sub-optimally. Typically, discrete optimization-based allocation and scheduling are performed during the pre-execution phase. To facilitate real-time replanning, relaxation methods can be employed alongside the previously mentioned problem scope constraints. Convex relaxation, commonly used in optimal control to guarantee optimality and manage computational costs,

as shown in [176–178], can be adapted to relax the nonlinear and discrete aspects of the task allocation problem. This adaptation helps limit computational expenses and enables replanning solutions during execution.

Furthermore, considering that agents may be dispersed at various locations when a failure occurs, a centralized planning approach may not be feasible. In such instances, as detailed in Sec. 6.3.1, a distributed planning framework can be utilized. This framework allows for the flexible generation of plans, taking into account communication constraints and the dynamic nature of agent locations, thereby enhancing the system's adaptability in the replanning stage.

6.3.3 Intra-team Hierarchical Task and Motion Planner

The intra-team trajectory planner discussed in Chapter 5 is currently limited to generating cooperative trajectories for short-term, simple tasks. For more complex, long-term missions, a higher-level planner is needed to integrate these learned simple maneuvers into comprehensive multi-agent tasks.

One potential approach is the planning domain definition languages (PDDL), which are widely used for planning sequential robotic tasks [73, 74, 179]. PDDL frames a task as a graph search problem, where the actions of robots form the edges of the graph, and the nodes represent states, defined as a conjunction of conditions describing the world. An action is executed based on certain preconditions and will change some conditions to reach the next state. Once the conditions and actions are defined, graph search methods can be employed to generate the optimal task plan from the initial state to the desired goal state. While PDDL-based methods do not explicitly model parallel asynchronous executions in multi-agent scenarios, there is scope to extend these methods for more complex multi-agent tasks.

We have recently seen the potential to use large language models (LLMs) to achieve long-horizon planning for multi-agent planning. As an example, the study in [180] uses an LLM to generate the action sequence for the agent to achieve long-horizon manipulation tasks.

BIBLIOGRAPHY

[1] J. Juett and B. Kuipers, "Learning to grasp by extending the peri-personal space graph," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 8695–8700.

[2] B. Fu, K. S. Shankar, and N. Michael, "Rad-vio: Rangefinder-aided downward visual-inertial odometry," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 1841–1847.

[3] M. Deng, X. Wang, D. Li, and C. C. Menassa, "Digital id framework for human-centric monitoring and control of smart buildings," in *Building Simulation*, vol. 15, no. 10. Springer, 2022, pp. 1709–1728.

[4] S. Park, X. Wang, C. C. Menassa, V. R. Kamat, and J. Y. Chai, "Natural language instructions for intuitive human interaction with robotic assistants in field construction work," *arXiv preprint arXiv:2307.04195*, 2023.

[5] X. Wang, H. Yu, W. McGee, C. C. Menassa, and V. R. Kamat, "Enabling bim-driven robotic construction workflows with closed-loop digital twins," *arXiv preprint arXiv:2306.09639*, 2023.

[6] S. Teng, D. Chen, W. Clark, and M. Ghaffari, "An error-state model predictive control on connected matrix lie groups for legged robot control," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 8850–8857.

[7] P. Mitrano, A. LaGrassa, O. Kroemer, and D. Berenson, "Focused adaptation of dynamics models for deformable object manipulation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5931–5937.

[8] J. Wilson, Y. Fu, A. Zhang, J. Song, A. Capodieci, P. Jayakumar, K. Barton, and M. Ghaffari, "Convolutional bayesian kernel inference for 3d semantic mapping," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 8364–8370.

[9] T. Toner, M. Saez, D. M. Tilbury, and K. Barton, "Opportunities and challenges in applying reinforcement learning to robotic manipulation: An industrial case study," *Manufacturing Letters*, vol. 35, pp. 1019–1030, 2023.

[10] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 272–11 281.

[11] J. Chen, J. Li, Y. Huang, C. Garrett, D. Sun, C. Fan, A. Hofmann, C. Mueller, S. Koenig, and B. C. Williams, "Cooperative task and motion planning for multi-arm assembly systems," *arXiv preprint arXiv:2203.02475*, 2022.

[12] C.-H. Chang, S.-C. Wang, and C.-C. Wang, "Exploiting moving objects: Multi-robot simultaneous localization and tracking," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 810–827, 2015.

[13] P. Tokekar, J. Vander Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic uav and ugv system for precision agriculture," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1498–1511, 2016.

[14] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, pp. 9–9, 2008.

[15] J. Werfel, K. Petersen, and R. Nagpal, "Designing collective behavior in a termite-inspired robot construction team," *Science*, vol. 343, no. 6172, pp. 754–758, 2014.

[16] D. Shishika, J. Paulos, and V. Kumar, "Cooperative team strategies for multi-player perimeter-defense games," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2738–2745, 2020.

[17] X. Cai, B. Schlotfeldt, K. Khosoussi, N. Atanasov, G. J. Pappas, and J. P. How, "Non-monotone energy-aware information gathering for heterogeneous robot teams," *arXiv preprint arXiv:2101.11093*, 2021.

[18] M. Quann, L. Ojeda, W. Smith, D. Rizzo, M. Castanier, and K. Barton, "An energy-efficient method for multi-robot reconnaissance in an unknown environment," in *Proceedings of the American Control Conference*. IEEE, 2017, pp. 2279–2284.

[19] ——, "Ground robot terrain mapping and energy prediction in environments with 3-d topography," in *Proceedings of the American Control Conference*. IEEE, 2018, pp. 3532–3537.

[20] ——, "Chance constrained reachability in environments with spatially varying energy costs," *Robotics and Autonomous Systems*, vol. 119, pp. 1–12, 2019.

[21] ——, "Power prediction for heterogeneous ground robots through spatial mapping and sharing of terrain data," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1579–1586, 2020.

[22] B. Schlotfeldt, V. Tzoumas, D. Thakur, and G. J. Pappas, "Resilient active information gathering with mobile robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2018, pp. 4309–4316.

[23] K. Yu, J. M. O'Kane, and P. Tokekar, "Coverage of an environment using energy-constrained unmanned aerial vehicles," in *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2019, pp. 3259–3265.

[24] Y. Sung, A. K. Budhiraja, R. K. Williams, and P. Tokekar, "Distributed assignment with limited communication for multi-robot multi-target tracking," *Autonomous Robots*, vol. 44, no. 1, pp. 57–73, 2020.

[25] Z. Xu and V. Tzoumas, "Resource-aware distributed submodular maximization: A paradigm for multi-robot decision-making," in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 5959–5966.

[26] Y. Tan and Z.-y. Zheng, "Research advance in swarm robotics," *Defence Technology*, vol. 9, no. 1, pp. 18–39, 2013.

[27] K. Sundar, S. Venkatachalam, and S. G. Manyam, "Path planning for multiple heterogeneous unmanned vehicles with uncertain service times," in *Proceedings of the International Conference on Unmanned Aircraft Systems*, 2017, pp. 480–487.

[28] L. Zhou and P. Tokekar, "Multi-robot coordination and planning in uncertain and adversarial environments," *Current Robotics Reports*, vol. 2, no. 2, pp. 147–157, 2021.

[29] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[30] A. Messing, G. Neville, S. Chernova, S. Hutchinson, and H. Ravichandar, "Grstaps: Graphically recursive simultaneous task allocation, planning, and scheduling," *The International Journal of Robotics Research*, vol. 41, no. 2, pp. 232–256, 2022.

[31] Y. Wang, J. Zhang, K. Assogba, Y. Liu, M. Xu, and Y. Wang, "Collaboration and transportation resource sharing in multiple centers vehicle routing optimization with delivery and pickup," *Knowledge-Based Systems*, vol. 160, pp. 296–310, 2018.

[32] A. Akbari, F. Lagriffoul, and J. Rosell, "Combined heuristic task and motion planning for bi-manual robots," *Autonomous robots*, vol. 43, no. 6, pp. 1575–1590, 2019.

[33] S. Thakar, A. Kabir, P. M. Bhatt, R. K. Malhan, P. Rajendran, B. C. Shah, and S. K. Gupta, "Task assignment and motion planning for bi-manual mobile manipulation," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2019, pp. 910–915.

[34] H. Zhang, S.-H. Chan, J. Zhong, J. Li, S. Koenig, and S. Nikolaidis, "A mip-based approach for multi-robot geometric task-and-motion planning," in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2022, pp. 2102–2109.

[35] I. Maza, K. Kondak, M. Bernard, and A. Ollero, "Multi-uav cooperation and control for load transportation and deployment," in *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009*. Springer, 2009, pp. 417–449.

[36] M. Corah, C. O'Meadhra, K. Goel, and N. Michael, "Communication-efficient planning and mapping for multi-robot exploration in large environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1715–1721, 2019.

[37] M. Turpin, N. Michael, and V. Kumar, "Decentralized formation control with variable shapes for aerial robots," in *2012 IEEE international conference on robotics and automation*. IEEE, 2012, pp. 23–30.

[38] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 11 785–11 792.

[39] J. Li, Z. Chen, Y. Zheng, S.-H. Chan, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig, "Scalable rail planning and replanning: Winning the 2020 flatland challenge," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 31, 2021, pp. 477–485.

[40] L. Zhou, V. Tzoumas, G. J. Pappas, and P. Tokekar, "Resilient active target tracking with multiple robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 1, pp. 129–136, 2018.

[41] E. Prianto, M. Kim, J.-H. Park, J.-H. Bae, and J.-S. Kim, "Path planning for multi-arm manipulators using deep reinforcement learning: Soft actor–critic with hindsight experience replay," *Sensors*, vol. 20, no. 20, p. 5911, 2020.

[42] E. Simetti, G. Casalino, F. Wanderlingh, and M. Aicardi, "A task priority approach to cooperative mobile manipulation: Theory and experiments," *Robotics and Autonomous Systems*, vol. 122, p. 103287, 2019.

[43] Y. He, M. Wu, and S. Liu, "A cooperative optimization strategy for distributed multi-robot manipulation with obstacle avoidance and internal performance maximization," *Mechatronics*, vol. 76, p. 102560, 2021.

[44] B. Fu, W. Smith, D. Rizzo, M. Castanier, and K. Barton, "Heterogeneous vehicle routing and teaming with gaussian distributed energy uncertainty," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2020, pp. 4315–4322.

[45] B. Fu, W. Smith, D. M. Rizzo, M. Castanier, M. Ghaffari, and K. Barton, "Robust task scheduling for heterogeneous robot teams under capability uncertainty," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 1087–1105, 2022.

[46] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[47] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[48] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," *Cooperative robots and sensor networks 2015*, pp. 31–51, 2015.

[49] E. Nunes, M. Manner, H. Mitiche, and M. Gini, "A taxonomy for task allocation problems with temporal and ordering constraints," *Robotics and Autonomous Systems*, vol. 90, pp. 55–70, 2017.

[50] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–31, 2019.

[51] A. Prorok, M. Malencia, L. Carlone, G. S. Sukhatme, B. M. Sadler, and V. Kumar, "Beyond robustness: A taxonomy of approaches towards resilient multi-robot systems," *arXiv preprint arXiv:2109.12343*, 2021.

[52] G. A. Korsah, B. Kannan, B. Browning, A. Stentz, and M. B. 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*. IEEE, 2012, pp. 115–122.

[53] C. Özgüven, Y. Yavuz, and L. Özbakır, "Mixed integer goal programming models for the flexible job-shop scheduling problems with separable and non-separable sequence dependent setup times," *Applied Mathematical Modelling*, vol. 36, no. 2, pp. 846–858, 2012.

[54] W.-Y. Ku and J. C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," *Computers & Operations Research*, vol. 73, pp. 165–173, 2016.

[55] J. Moser, J. Hoffman, R. Hildebrand, and E. Komendera, "A flexible job shop scheduling representation of the autonomous in-space assembly task assignment problem," *arXiv preprint arXiv:2003.12148*, 2020.

[56] S. Liemhetcharat and M. Veloso, "Modeling mutual capabilities in heterogeneous teams for role assignment," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 3638–3644.

[57] ——, "Weighted synergy graphs for role assignment in ad hoc heterogeneous robot teams," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5247–5254.

[58] E. G. Jones, M. B. Dias, and A. Stentz, "Time-extended multi-robot coordination for domains with intra-path constraints," *Autonomous Robots*, vol. 30, no. 1, pp. 41–56, 2011.

[59] M. Pujol-Gonzalez, J. Cerquides, P. Meseguer, and J. A. Rodríguez-Aguilar, "Efficient inter-team task allocation in robocup rescue," *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2015.

[60] D. Song, K. Huebner, V. Kyrki, and D. Kragic, "Learning task constraints for robot grasping using graphical models," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 1579–1585.

[61] C. H. Ek, D. Song, K. Huebner, and D. Kragic, "Task modeling in imitation learning using latent variable models," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2010, pp. 548–553.

[62] R. Zlot and A. Stentz, "Multirobot control using task abstraction in a market framework," in *Collaborative Technology Alliances Conference*, 2003.

[63] ——, "Market-based multirobot coordination using task abstraction," in *Field and Service Robotics*. Springer, 2003, pp. 167–177.

[64] ——, "Complex task allocation for multiple robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2005, pp. 1515–1522.

[65] B. Hayes and B. Scassellati, "Autonomously constructing hierarchical task networks for planning and human-robot collaboration," in *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2016, pp. 5469–5476.

[66] S. D. Klee, G. Gemignani, D. Nardi, and M. Veloso, "Graph-based task libraries for robots: Generalization and autocompletion," in *Congress of the Italian Association for Artificial Intelligence*. Springer, 2015, pp. 397–409.

[67] M. N. Nicolescu and M. J. Mataric, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 2003, pp. 241–248.

[68] S. Ekvall and D. Kragic, "Robot learning from demonstration: a task-level planning approach," *International Journal of Advanced Robotic Systems*, vol. 5, no. 3, p. 33, 2008.

[69] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto, "Learning and generalization of complex tasks from unstructured demonstrations," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5239–5246.

[70] D. H. Grollman and O. C. Jenkins, "Incremental learning of subtasks from unsegmented demonstration," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 261–266.

[71] B. Hayes and B. Scassellati, "Effective robot teammate behaviors for supporting sequential manipulation tasks," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2015, pp. 6374–6380.

[72] C. Galindo, J.-A. Fernández-Madrigal, J. González, and A. Saffiotti, "Robot task planning using semantic maps," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 955–966, 2008.

[73] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. SRI, A. Barrett, D. Christianson *et al.*, "Pddl| the planning domain definition language," *Technical Report*, 1998.

[74] A. Torreño, E. Onaindia, A. Komenda, and M. Štolba, "Cooperative multi-agent planning: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–32, 2017.

[75] J. Benton, A. Coles, and A. Coles, "Temporal planning with preferences and time-dependent continuous costs," in *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[76] O. Sapena, A. Torreno, and E. Onaindía, "Parallel heuristic search in forward partial-order planning," *The Knowledge Engineering Review*, vol. 31, no. 5, pp. 417–428, 2016.

[77] Y. Carreno, È. Pairet, Y. Petillot, and R. P. Petrick, "A decentralised strategy for heterogeneous auv missions via goal distribution and temporal planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 431–439.

[78] M. Crosby and R. P. Petrick, "Temporal multiagent planning with concurrent action constraints," in *ICAPS workshop on distributed and multi-agent planning (DMAP)*. Citeseer, 2014.

[79] M. Cashmore, A. Coles, B. Cserna, E. Karpas, D. Magazzeni, and W. Ruml, "Temporal planning while the clock ticks," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, no. 1, 2018.

[80] F. Faruq, D. Parker, B. Lacerda, and N. Hawes, "Simultaneous task allocation and planning under uncertainty," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2018, pp. 3559–3564.

[81] H. Ravichandar, K. Shaw, and S. Chernova, "Strata: unified framework for task assignments in large teams of heterogeneous agents," *Autonomous Agents and Multi-Agent Systems*, vol. 34, no. 2, 2020.

[82] M. Rudolph, S. Chernova, and H. Ravichandar, "Desperate times call for desperate measures: Towards risk-adaptive task allocation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 2592–2597.

[83] S. Sarykalin, G. Serraino, and S. Uryasev, "Value-at-risk vs. conditional value-at-risk in risk management and optimization," in *State-of-the-art decision-making tools in the information-intensive age*. Informs, 2008, pp. 270–294.

[84] A. Majumdar and M. Pavone, "How should a robot assess risk? towards an axiomatic theory of risk in robotics," in *Robotics Research*. Springer, 2020, pp. 75–84.

[85] R. Balasubramanian, L. Zhou, P. Tokekar, and P. Sujit, "Risk-aware submodular optimization for multi-objective travelling salesperson problem," *arXiv preprint arXiv:2011.01095*, 2020.

[86] J. Bernhard, S. Pollok, and A. Knoll, "Addressing inherent uncertainty: Risk-sensitive behavior generation for automated driving using distributional reinforcement learning," in *Proceedings of the IEEE Intelligent Vehicle Symposium*. IEEE, 2019, pp. 2148–2155.

[87] L. Lindemann, G. J. Pappas, and D. V. Dimarogonas, "Control barrier functions for nonholonomic systems under risk signal temporal logic specifications," in *Proceedings of the IEEE Conference on Decision and Control*. IEEE, 2020, pp. 1422–1428.

[88] A. Hakobyan, G. C. Kim, and I. Yang, "Risk-aware motion planning and control using cvar-constrained optimization," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3924–3931, 2019.

[89] L. Zhou and P. Tokekar, "An approximation algorithm for risk-averse submodular optimization," in *International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2018, pp. 144–159.

[90] M. Koes, I. Nourbakhsh, K. Sycara, M. Koes, K. Sycara, I. Nourbakhsh, M. Koes, I. Nourbakhsh, K. Sycara, S. D. Ramchurn *et al.*, "Heterogeneous multirobot coordination with spatial and temporal constraints," in *Proceedings of the AAAI National Conference on Artificial Intelligence*, vol. 5, 2005, pp. 1292–1297.

[91] S. D. Ramchurn, M. Polukarov, A. Farinelli, N. Jennings, and C. Trong, "Coalition formation with spatial and temporal constraints," *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2010.

[92] L. Vig and J. A. Adams, "Multi-robot coalition formation," *IEEE transactions on robotics*, vol. 22, no. 4, pp. 637–649, 2006.

[93] G. Neville, A. Messing, H. Ravichandar, S. Hutchinson, and S. Chernova, "An interleaved approach to trait-based task allocation and scheduling," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1507–1514.

[94] A. Prorok, M. A. Hsieh, and V. Kumar, "The impact of diversity on optimal control policies for heterogeneous robot swarms," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 346–358, 2017.

[95] B. Fu, T. Kathuria, D. Rizzo, M. Castanier, X. J. Yang, M. Ghaffari, and K. Barton, "Simultaneous human-robot matching and routing for multi-robot tour guiding under time uncertainty," *Journal of Autonomous Vehicles and Systems*, vol. 1, no. 4, p. 041005, 2021.

[96] ——, "Human-robot matching and routing for multi-robot tour guiding under time uncertainty," *arXiv preprint arXiv:2309.15373*, 2023.

[97] R. T. Rockafellar, S. Uryasev *et al.*, "Optimization of conditional value-at-risk," *Journal of risk*, vol. 2, pp. 21–42, 2000.

[98] S. Asmussen and P. W. Glynn, *Stochastic simulation: algorithms and analysis*. Springer Science & Business Media, 2007, vol. 57.

[99] J. R. Birge and F. Louveaux, *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[100] T. Barfoot, J. Burgner-Kahrs, E. Diller, A. Garg, A. Goldenberg, J. Kelly, X. Liu, H. Naguib, G. Nejat, A. Schoellig *et al.*, "Making sense of the robotized pandemic response: a comparison of global and canadian robot deployments and success factors," *arXiv preprint arXiv:2009.08577*, 2020.

[101] E. Olson, J. Leonard, and S. Teller, "Fast iterative alignment of pose graphs with poor initial estimates," in *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2006, pp. 2262–2269.

[102] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006, vol. 2, no. 3.

[103] I. Pohl, "The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1973, pp. 12–17.

[104] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*. Springer Science & Business Media, 2003, vol. 24.

[105] I. Heller and C. Tompkins, "An extension of a theorem of dantzig's," *Linear inequalities and related systems*, vol. 38, pp. 247–254, 1956.

[106] S. Satake, T. Kanda, D. F. Glas, M. Imai, H. Ishiguro, and N. Hagita, "How to approach humans? strategies for social robots to initiate interaction," in *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*. San Diego, CA: IEEE, 2009, pp. 109–116.

[107] M. Shiomi, T. Kanda, D. F. Glas, S. Satake, H. Ishiguro, and N. Hagita, "Field trial of networked social robots in a shopping mall," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 2846–2853.

[108] M. Doering, T. Kanda, and H. Ishiguro, "Neural-network-based memory for a social robot: Learning a memory model of human behavior from data," *ACM Transactions on Human-Robot Interaction (THRI)*, vol. 8, no. 4, pp. 1–27, 2019.

[109] R. Kirby, J. Forlizzi, and R. Simmons, "Affective social robots," *Robotics and Autonomous Systems*, vol. 58, no. 3, pp. 322–332, 2010.

[110] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The interactive museum tour-guide robot," in *Proceedings of the AAAI National Conference on Artificial Intelligence*, USA, 1998, p. 11–18.

[111] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The museum tour-guide robot rhino," in *Autonome Mobile Systeme 1998*. Springer, 1999, pp. 245–254.

[112] B. De Carolis, G. Palestra, C. Della Penna, M. Cianciotta, and A. Cervelione, "Social robots supporting the inclusion of unaccompanied migrant children: Teaching the meaning of culture-related gestures," *Journal of e-Learning and Knowledge Society*, vol. 15, no. 2, pp. 43–57, 2019.

[113] Y. Kurata, "Interactive assistance for tour planning," in *Spatial Cognition VII*, C. Hölscher, T. F. Shipley, M. Olivetti Belardinelli, J. A. Bateman, and N. S. Newcombe, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 289–302.

[114] M. Bennewitz, F. Faber, D. Joho, M. Schreiber, and S. Behnke, "Towards a humanoid museum guide robot that interacts with multiple persons," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2005, pp. 418–423.

[115] L. R. Ford and D. R. Fulkerson, *Flows in networks*. Princeton university press, 2015.

[116] Y.-Q. Cheng, V. Wu, R. Collins, A. R. Hanson, and E. M. Riseman, "Maximum-weight bipartite matching technique and its application in image feature matching," in *Visual Communications and Image Processing*, vol. 2727. International Society for Optics and Photonics, 1996, pp. 453–462.

[117] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Proceedings of the European Conference on Computer Vision*. Springer, 2020, pp. 213–229.

[118] G. Aggarwal, G. Goel, C. Karande, and A. Mehta, "Online vertex-weighted bipartite matching and single-bid budgeted allocations," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2011, pp. 1253–1264.

[119] A. Dutta and A. Asaithambi, "One-to-many bipartite matching based coalition formation for multi-robot task allocation," in *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2019, pp. 2181–2187.

[120] J. E. Mendoza and J. G. Villegas, "A multi-space sampling heuristic for the vehicle routing problem with stochastic demands," *Optimization Letters*, vol. 7, no. 7, pp. 1503–1516, 2013.

[121] N. Secomandi and F. Margot, "Reoptimization approaches for the vehicle-routing problem with stochastic demands," *Operations research*, vol. 57, no. 1, pp. 214–230, 2009.

[122] Y. Marinakis, G.-R. Iordanidou, and M. Marinaki, "Particle swarm optimization for the vehicle routing problem with stochastic demands," *Applied Soft Computing*, vol. 13, no. 4, pp. 1693–1704, 2013.

[123] L. Chen, M. H. Hà, A. Langevin, and M. Gendreau, "Optimizing road network daily maintenance operations with stochastic service and travel times," *Transportation Research Part E: Logistics and Transportation Review*, vol. 64, pp. 88–102, 2014.

[124] X. Li, P. Tian, and S. C. Leung, "Vehicle routing problems with time windows and stochastic travel and service times: Models and algorithm," *International Journal of Production Economics*, vol. 125, no. 1, pp. 137–145, 2010.

[125] A. Gómez, R. Mariño, R. Akhavan-Tabatabaei, A. L. Medaglia, and J. E. Mendoza, "On modeling stochastic travel and service times in vehicle routing," *Transportation Science*, vol. 50, no. 2, pp. 627–641, 2016.

[126] S. Venkatachalam, M. Bansal, J. M. Smereka, and J. Lee, "Two-stage stochastic programming approach for path planning problems under travel time and availability uncertainties," *arXiv preprint arXiv:1910.04251*, 2019.

[127] S. Venkatachalam, K. Sundar, and S. Rathinam, "A two-stage approach for routing multiple unmanned aerial vehicles with stochastic fuel consumption," *Sensors*, vol. 18, no. 11, p. 3756, 2018.

[128] P. Toth and D. Vigo, *The vehicle routing problem*. SIAM, 2002.

[129] M. Schreieck, H. Safetli, S. A. Siddiqui, C. Pflügler, M. Wiesche, and H. Krcmar, "A matching algorithm for dynamic ridesharing," *Transportation Research Procedia*, vol. 19, pp. 272–285, 2016.

[130] O. F. Aydin, I. Gokasar, and O. Kalan, "Matching algorithm for improving ride-sharing by incorporating route splits and social factors," *PloS one*, vol. 15, no. 3, p. e0229674, 2020.

[131] P. Sitek and J. Wikarek, "Capacitated vehicle routing problem with pick-up and alternative delivery (cvrppad): model and implementation using hybrid approach," *Annals of Operations Research*, vol. 273, no. 1, pp. 257–277, 2019.

[132] L. He, X. Liu, G. Laporte, Y. Chen, and Y. Chen, "An improved adaptive large neighborhood search algorithm for multiple agile satellites scheduling," *Computers & Operations Research*, vol. 100, pp. 12–25, 2018.

[133] M. Alinaghian and N. Shokouhi, "Multi-depot multi-compartment vehicle routing problem, solved by a hybrid adaptive large neighborhood search," *Omega*, vol. 76, pp. 85–99, 2018.

[134] M. Deng, B. Fu, and C. Menassa, "Room match: Achieving thermal comfort through smart space allocation and environmental control in buildings," in *2021 Winter Simulation Conference (WSC)*. IEEE, 2021, pp. 1–11.

[135] M. Deng, B. Fu, C. C. Menassa, and V. R. Kamat, "Learning-based personal models for joint optimization of thermal comfort and energy consumption in flexible workplaces," *Energy and Buildings*, vol. 298, p. 113438, 2023.

[136] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[137] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A platform for embodied ai research," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9339–9347.

[138] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," *arXiv preprint arXiv:1709.06158*, 2017.

[139] A. Kadian, J. Truong, A. Gokaslan, A. Clegg, E. Wijmans, S. Lee, M. Savva, S. Chernova, and D. Batra, "Sim2real predictivity: Does evaluation in simulation predict real-world performance?" *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6670–6677, 2020.

[140] A. Srikanthan and H. Ravichandar, "Resource-aware adaptation of heterogeneous strategies for coalition formation," in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 1732–1734.

[141] B. Fu, W. Smith, D. Rizzo, M. Castanier, M. Ghaffari, and K. Barton, "Learning task requirements and agent capabilities for multi-agent task allocation," *arXiv preprint arXiv:2211.03286*, 2022.

[142] C. Banks, S. Wilson, S. Coogan, and M. Egerstedt, "Multi-agent task allocation using cross-entropy temporal logic optimization," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7712–7718.

[143] D. Aksaray, K. Leahy, and C. Belta, "Distributed multi-agent persistent surveillance under temporal logic constraints," *IFAC-PapersOnLine*, vol. 48, no. 22, pp. 174–179, 2015.

[144] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," *arXiv preprint arXiv:1909.07528*, 2019.

[145] A. Elfakharany, R. Yusof, and Z. Ismail, "Towards multi robot task allocation and navigation using deep reinforcement learning," in *Journal of Physics: Conference Series*, vol. 1447, no. 1. IOP Publishing, 2020, p. 012045.

[146] B. Wang, Z. Liu, Q. Li, and A. Prorok, "Mobile robot path planning in dynamic environments through globally guided reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6932–6939, 2020.

[147] H. Wu, A. Ghadami, A. E. Bayrak, J. M. Smereka, and B. I. Epureanu, "Impact of heterogeneity and risk aversion on task allocation in multi-agent teams," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7065–7072, 2021.

[148] G. Chou, N. Ozay, and D. Berenson, "Explaining multi-stage tasks by learning temporal logic formulas from suboptimal demonstrations," *arXiv preprint arXiv:2006.02411*, 2020.

[149] M. Vazquez-Chanlatte, S. Jha, A. Tiwari, M. K. Ho, and S. Seshia, "Learning task specifications from demonstrations," *Advances in neural information processing systems*, vol. 31, 2018.

[150] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, no. 1, 2019, pp. 151–158.

[151] H. Ma, J. Li, T. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," *arXiv preprint arXiv:1705.10868*, 2017.

[152] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 7643–7650.

[153] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Mapf-lns2: fast repairing for multi-agent path finding via large neighborhood search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 9, 2022, pp. 10 256–10 265.

[154] F. Krebs and T. Asfour, "A bimanual manipulation taxonomy," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 031–11 038, 2022.

[155] D. Rakita, B. Mutlu, M. Gleicher, and L. M. Hiatt, "Shared control–based bimanual robot manipulation," *Science Robotics*, vol. 4, no. 30, p. eaaw0955, 2019.

[156] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 7512–7519.

[157] L. Zhou, V. D. Sharma, Q. Li, A. Prorok, A. Ribeiro, P. Tokekar, and V. Kumar, "Graph neural networks for decentralized multi-robot target tracking," in *2022 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2022, pp. 195–202.

[158] M. Bettini, A. Shankar, and A. Prorok, "Heterogeneous multi-robot reinforcement learning," *arXiv preprint arXiv:2301.07137*, 2023.

[159] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, pp. 1–49, 2022.

[160] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

[161] C. Wu, A. Rajeswaran, Y. Duan, V. Kumar, A. M. Bayen, S. Kakade, I. Mordatch, and P. Abbeel, "Variance reduction for policy gradient with action-dependent factorized baselines," *arXiv preprint arXiv:1803.07246*, 2018.

[162] P. Mitrano, D. McConachie, and D. Berenson, "Learning where to trust unreliable models in an unstructured world for deformable object manipulation," *Science Robotics*, vol. 6, no. 54, p. eabd8170, 2021.

[163] Y. Chen, T. Wu, S. Wang, X. Feng, J. Jiang, Z. Lu, S. McAleer, H. Dong, S.-C. Zhu, and Y. Yang, "Towards human-level bimanual dexterous manipulation with reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 5150–5163, 2022.

[164] A. Petrenko, A. Allshire, G. State, A. Handa, and V. Makoviychuk, "Dexpbt: Scaling up dexterous manipulation for hand-arm systems with population based training," *arXiv preprint arXiv:2305.12127*, 2023.

[165] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

[166] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE transactions on robotics*, vol. 25, no. 4, pp. 912–926, 2009.

[167] H.-L. Choi, A. K. Whitten, and J. P. How, "Decentralized task allocation for heterogeneous teams with cooperation constraints," in *Proceedings of the 2010 American Control Conference*. IEEE, 2010, pp. 3057–3062.

[168] A. K. Whitten, H.-L. Choi, L. B. Johnson, and J. P. How, "Decentralized task allocation with coupled constraints in complex missions," in *Proceedings of the 2011 American Control Conference*. IEEE, 2011, pp. 1642–1649.

[169] S. Hunt, Q. Meng, C. Hinde, and T. Huang, "A consensus-based grouping algorithm for multi-agent cooperative task allocation with complex requirements," *Cognitive computation*, vol. 6, no. 3, pp. 338–350, 2014.

[170] G. Ninikas and I. Minis, "Replanning in vehicle routing with dynamic pickups," *Technical report*, 2014.

[171] A. Schoenig and M. Pagnucco, "Evaluating sequential single-item auctions for dynamic task allocation," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2010, pp. 506–515.

[172] J. Oyola, H. Arntzen, and D. L. Woodruff, "The stochastic vehicle routing problem, a literature review, part ii: solution methods," *EURO Journal on Transportation and Logistics*, vol. 6, no. 4, pp. 349–388, 2017.

[173] ——, "The stochastic vehicle routing problem, a literature review, part i: models," *EURO Journal on Transportation and Logistics*, vol. 7, no. 3, pp. 193–221, 2018.

[174] G. Laporte, F. V. Louveaux, and L. Van Hamme, "An integer l-shaped algorithm for the capacitated vehicle routing problem with stochastic demands," *Operations Research*, vol. 50, no. 3, pp. 415–423, 2002.

[175] S. S. Fazeli, S. Venkatachalam, R. B. Chinnam, and A. Murat, "Two-stage stochastic choice modeling approach for electric vehicle charging station network design in urban communities," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 5, pp. 3038–3053, 2020.

[176] P. Zhao, S. Mohan, and R. Vasudevan, "Control synthesis for nonlinear optimal control via convex relaxations," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 2654–2661.

[177] ——, "Optimal control for nonlinear hybrid systems via convex relaxations," *arXiv preprint arXiv:1702.04310*, 2017.

[178] G. Fazelnia, R. Madani, A. Kalbat, and J. Lavaei, "Convex relaxation for optimal distributed control problems," *IEEE Transactions on Automatic Control*, vol. 62, no. 1, pp. 206–221, 2016.

[179] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[180] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg, "Text2motion: From natural language instructions to feasible plans," *arXiv preprint arXiv:2303.12153*, 2023.