HETEROGENEITY- AND RISK-AWARE ALGORITHMS FOR TASK ALLOCATION TO MOBILE AGENTS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Amritha Prasad

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Shreyas Sundaram, Chair

School of Electrical and Computer Engineering, Purdue University

Dr. Jianghai Hu

School of Electrical and Computer Engineering, Purdue University

Dr. Shaoshuai Mou

School of Aeronautics and Astronautics, Purdue University

Dr. Richard M. Voyles

School of Engineering Technology, Purdue University

Dr. Jeffrey Hudack

Air Force Research Laboratory

Approved by:

Dr. Dimitrios Peroulis

Head of the School Graduate Program

To my parents.

Thank you for believing in me.

I couldn't have done it without you.

ACKNOWLEDGMENTS

I would like to express my appreciation and gratitude to Professor Shreyas Sundaram, my Major Professor, for his support and guidance throughout my time at Purdue. He has been an exceptional mentor and I consider it a privilege to have worked with him. Without his help, this thesis would not have been possible. The skills and values that Professor Sundaram has helped me learn will help me throughout my career and life.

I would like to acknowledge the valuable inputs I have received from the members of my Doctoral Committee, Professor Jianghai Hu, Professor Shaoshuai Mou, Professor Richard Voyles and Dr. Jeffrey Hudack. I would like to express my gratitude to Dr. Sita Ambarao for her encouragement and support. She has been a valuable mentor and a constant source of inspiration.

My heartfelt thanks to my family, in particular my parents and husband, for being constantly supportive, always encouraging me to pursue bigger goals and sharing my hopes and dreams.

I also thank my lab-mates, both former and current, for their help with research, lively discussions and company. Lastly but not the least, I would like to thank all my teachers through the years, without whose help and encouragement, I would never have reached here.

TABLE OF CONTENTS

Page

LIST OF FIGURES vii

ABSTRACT viii

1 INTRODUCTION 1

1.1 Contributions 3

1.2 Outline 4

2 BACKGROUND 6

2.1 Multi-agent Systems 6

2.2 Traveling Salesperson Problem (TSP) 7

2.3 Objectives of Routing Problems 9

2.4 Related Problems in the Literature 10

2.5 Complexity Theory 10

2.6 Assumptions 11

3 MIN-MAX TOURS FOR TASK ALLOCATION TO HETEROGENEOUS AGENTS 12

3.1 Problem Statement 15

3.2 A Naive Algorithm for HACP 16

3.3 Constant Factor Approximation Algorithms for HACP 18

3.3.1 Cycle Splitting Approach 18

3.3.2 Min-Max Splitting Approach for Heterogeneous Agents 21

3.3.3 Examples Illustrating the Algorithms 27

3.4 Chapter Summary 31

4 MIN-MAX PATHS FOR TASK ALLOCATION TO HETEROGENEOUS AGENTS 32

4.1 Problem Statement 32

Page

4.2 Approach 1: Extending MMPCP to HAPP . . . 34
4.3 Approach 2: Solution to HAPP Based on Trees . . . 37
4.3.1 Constant Factor Algorithm for HATP . . . 40
4.3.2 Using HATP to solve HAPP . . . 46
4.4 Comparison of the approaches to solve HAPP . . . 48
4.5 Simulations . . . 49
4.6 Chapter Summary . . . 51
5 OPTIMAL POLICIES FOR RISK-AWARE SENSOR DATA COLLECTION BY A MOBILE AGENT . . . 53
5.1 Problem Formulation and Notation . . . 54
5.1.1 Cost of Loss of Agent . . . 56
5.2 Characteristics of the Optimal Walks . . . 57
5.2.1 Ordering of the Cycles . . . 58
5.2.2 Skipping Tasks . . . 60
5.2.3 Bound on Length of Cycles . . . 63
5.3 Optimal Walks for Sufficiently Small ψ . . . 65
5.4 Evaluation of Heuristics for SARATE . . . 66
5.5 Chapter Summary . . . 70
6 POLICIES FOR RISK-AWARE SENSOR DATA COLLECTION BY HETEROGENEOUS MOBILE AGENTS . . . 71
6.1 Problem Formulation and Notation . . . 71
6.1.1 Submodularity of the Agent Utilities in High-Risk Scenarios . . . 74
6.1.2 Approximation Algorithms . . . 81
6.2 Simulations . . . 85
6.3 Chapter Summary . . . 88
7 SUMMARY AND FUTURE WORK . . . 89
7.1 Future Directions . . . 90
REFERENCES . . . 91
VITA . . . 96

LIST OF FIGURES

Figure Page

3.1 Task locations in Example 3.3.5 28

3.2 Task locations in Example 3.3.6 29

3.3 Task allocation in Example 3.3.7 30

4.1 Bipartite graph $H[X, Y]$ with the root nodes in partition X and the sub-trees in partition Y 45

4.2 Two solution approaches to HAPP 49

4.3 Output of Algorithm 5 on an instance of HAPP with 6 agents and 100 tasks.50

4.4 Run time of Algorithm 5 for various sizes of problem instances. 51

5.1 Task locations in Example 5.2.1 62

5.2 Comparison of performance of different heuristic algorithms: (a) Expected utility when tasks have homogeneous rewards (50) versus heterogeneous rewards (between 1 and 99), (b) Expected utility as the fraction of high reward tasks vary, (c) Run time as the number of tasks vary. 67

5.3 Comparison of performance of different algorithms to the optimal (brute force) solution. 70

6.1 Task locations in Example 6.1.7 80

6.2 Comparison of performance of greedy allocation and the optimal allocation when the number of agents is two. 87

6.3 Comparison of performance of greedy allocation and the optimal allocation when the number of agents is three. 87

6.4 Comparison of performance of greedy allocation and the optimal allocation when the number of agents is four. 88

ABSTRACT

Prasad, Amritha Ph.D., Purdue University, August 2020. Heterogeneity- and Risk-Aware Algorithms for Task Allocation To Mobile Agents. Major Professor: Shreyas Sundaram.

In this thesis, we investigate and characterize policies for task allocation to teams of agents in settings with heterogeneity and risk. We first consider a scenario consisting of a set of heterogeneous mobile agents located at a base (or depot), and a set of tasks dispersed over a geographic area. The agents are partitioned into different types. The tasks are partitioned into specialized tasks that can only be done by agents of a certain type, and generic tasks that can be done by any agent. The distances between every pair of tasks are specified, and satisfy the triangle inequality. Given this scenario, we address the problem of allocating these tasks among the available agents (subject to type compatibility constraints) while minimizing the maximum travel cost for any agent. We first look at the Heterogeneous Agent Cycle Problem (HACP) where agents start at a common base (or depot) and need to tour the set of tasks allocated to them before returning to the base. This problem is NP-hard, and we give a three phase algorithm to solve this problem that provides 5-factor approximation, regardless of the total number of agents and the number of agents of each type. We also show that in the special case where there is only one agent of each type, the algorithm has an approximation factor of 4. We then consider the Heterogeneous Agent Path Problem (HAPP) where agents can start from arbitrary locations and are not constrained to return to their start location. We consider two approaches to solve HAPP, and provide a 15-approximation algorithm for HAPP.

We then look at the effect of risk on path planning by considering a scenario where a mobile agent is required to collect measurements from a geographically dispersed

set of sensors and return them to a base. The agent faces a risk of destruction while traversing the environment to reach the sensors, and gets the reward for gathering a sensor measurement only if it successfully returns to base. We call this the Single Agent Risk Aware Task Execution (SARATE) problem. We characterize several properties of the optimal policy for the agent based on a quantity that we term the "reward-to-risk" ratio of the tasks. We provide the optimal policy when the risk of destruction is sufficiently high, and evaluate several heuristic policies via simulation.

Lastly, we extend our analysis of risk-aware path planning to scenarios with multiple agents. Our analysis allows for agents to be heterogeneous. We show that the scoring scheme is submodular when the risk is sufficiently high, and greedy algorithm gives solutions that provide a utility that is guaranteed to be within 50% of the optimal utility.

1. INTRODUCTION

The reliance of humankind on robots and artificial intelligence is increasing at a rapid pace. More processes are getting automated and we are progressively offloading more tasks from humans to machines, saving time, effort and money. Some of the common instances that we see in our every day life includes self-service kiosks, material handling systems, and, increasingly, self-driving cars. In applications pertaining to environmental monitoring, reconnaissance, and search-and-rescue, there is a general shift in focus from a single agent framework to a multi-agent framework to improve robustness, reduce complexity and execution time. A variety of such problems have been studied in the literature [1–4]. In such multi-agent systems, the agents cooperatively work towards accomplishing a common goal. The presence of multiple agents has several advantages. More tasks can be executed in parallel resulting in faster completion. Furthermore, the failure of one agent does not affect other agents (and the remaining agents may still achieve the goal, albeit with degraded performance). This is particularly useful in routing problems, where agents collectively work to minimize the delay in accomplishing the goal (or completing the overall task). This is very important in applications that are time critical or where the quality of service is defined by latency. Our focus in this thesis is on this class of routing problems, and thus all agents that we consider will be taken to be mobile agents.

Multi-agent systems may be composed of homogeneous agents or heterogeneous agents, i.e., the agents in the system may be similar to each other, or the agents in the system could be dissimilar. One way the agents could be dissimilar is when the constraints on motion of agents are different [5, 6]. Another form of heterogeneity is one where the agents differ in their functionality. For instance, one agent may have sensors mounted on it, while another agent might have the capacity to carry loads. Such heterogeneity in agents is an interesting area to study, primarily because

a variety of real world applications involve multiple requirements. For instance, an exploration mission would require functions like collecting samples and returning to the base, taking sensor measurements, imaging, etc. If there are multiple instances of each of these types of requirements, there arises the problem of how to efficiently route the available agents to all the required locations. In particular, the routing must satisfy the functional constraint that the tasks in an agent's route must be a type of task that the agent is able to fulfill. This motivates the study of task allocation to heterogeneous agents and finding routes for these agents to traverse.

Routing problems (or path planning problems) consists of the general class of problems where given a set of locations that are to be visited, the objective is to find the optimal route (or path) for an agent to traverse. There are several variants of this problem studied in the literature. The variations may be due to the number of agents present, the objective (or the definition of what is "optimal") or the constraints involved (such as time windows, or carrying capacity). In Chapter 2, we provide a brief summary of various routing and path planning problems.

Another area of interest is to study these routing problems in the presence of risk. That is, situations where there exists a chance that the agent may fail (or is lost) while traveling or executing a task. Most real-time applications have such risks associated with them - either inherent to the agent (for example, a system malfunction) or due to environmental hazards (such as fire or other accidents). This is particularly important when we are operating in hazardous or hostile environments. Some of the interesting questions to ask here are the following. In the presence of risk, should the agents take the same route as they did when there was no risk? How does the allocation of tasks to an agent change under risk? Does the order in which tasks are executed matter? In this thesis, we first consider the single agent version of this problem and look at some of these questions, and then extend our analysis to the multi-agent case.

1.1 Contributions

A majority of the work on the Traveling Salesperson Problem (TSP) and other routing problems minimize the total distance of the tours. This metric is useful, for instance, when trying to minimize the fuel cost (or the miles a vehicle is driven). Another important metric is the latency. In case of critical missions, such as search and rescue, one wishes to have minimal latency. To address this class of problems, we study the Heterogeneous Agent Cycle Problem (HACP), where the objective is to minimize the maximum cost among all agents, thus lowering the overall completion time, while accounting for functional heterogeneity among the agents. While several works in the literature looks at heterogeneous agents, a majority of them focuses on heterogeneity in terms of differences in vehicle dynamics (or constraints on vehicle motion). We provide algorithms to solve the Heterogeneous Agent Cycle Problem and provide guarantees that the solution obtained is no worse than a factor of 5 times the "optimal" solution, independent of the number of agents available or the number of tasks to be executed. We also show that in the special case where all agents are distinct, this factor reduces to 4.

We then generalize HACP to a problem we call the Heterogeneous Agent Path Problem (HAPP). In this problem, we relax some of the restrictions that were present in HACP, i.e., the agent's start locations are allowed to be different (they are not restricted to start from the base). Furthermore, since the start locations of agents can be arbitrary, there is no requirement for the agents to rendezvous back to their start locations. Thus, in HAPP, we find paths for agents to traverse instead of tours. This framework allows for adaptive allocation of tasks to heterogeneous agents. When new tasks arise, we can formulate a version of HAPP based on the current location of agents and tasks and find a path for each agent to traverse. We provide two approaches to solve this problem. In the first approach, we extend the work on path planning for homogeneous agents to heterogeneous agents. We prove that, using this approach, we can get solutions within a factor 15 of the optimal solution. In the

second approach, we first consider a problem we call the Heterogeneous Agent Tree Problem (HATP). In this problem, we first find trees for agents to traverse such that all the tasks are completed (subject to agent-task compatibility constraints). We then use the solution of HATP to find a solution for HAPP. We show that our algorithm using this method provides solutions that perform within a factor 16 of the optimal solution.

The above two problems study the path planning problem when there is no risk to any agent. Our third contribution is to study the Single Agent Risk Aware Task Execution (SARATE) problem, where we take into consideration the risk an agent faces. We prove several key properties of the optimal policy. Specifically, for the case where the risk is sufficiently high, we provide the optimal policy and show that it can be found in time that is polynomially related to the size of the input. We also provide some heuristics and evaluate their performance via simulations. Our fourth contribution is to extend the study of risk-aware path planning to multiple heterogeneous agents. We formulate the Multi-Agent Risk-Aware Task Allocation (MARATA) problem, by extending the SARATE formulation to multiple agents. In particular, we show that under sufficiently high-risk scenarios, the scoring scheme is submodular and a greedy algorithm provides a solution that is guaranteed to be within 50% of the optimal utility.

1.2 Outline

In Chapter 2, we provide some background on the traveling salesperson and vehicle routing problems. In Chapter 3, we focus on a specific problem called the Heterogeneous Agent Cycle Problem (HACP), where we consider a framework consisting of a set of heterogeneous agents located at a base that are required to collectively complete a set of tasks dispersed over a geographic area and then return to the base. Tours are found for agents so that they collectively complete all the tasks and the maximum cost among all the agents to tour its tasks is minimized. In Chapter 4, we

consider the generalization of HACP called the Heterogeneous Agent Path Problem (HAPP), which caters to resilient and adaptive allocation of tasks to heterogeneous agents. In HAPP, the agents can start from any arbitrary location and are not required to rendezvous back to their start location. Then, in Chapter 5, we consider the effect of risk on task execution. In particular, we study the Single Agent Risk Aware Task Execution (SARATE) problem. In this case, the agent aims to collect sensor measurements dispersed over a geographic area, but faces a non-zero risk of failing while it is traveling. In Chapter 6, we extend our analysis of risk-aware task allocation to multiple agents. We consider the scenario where multiple agents (that may be heterogeneous) are available to collect the sensor measurements, and study the Multi-Agent Risk-Aware Task Allocation (MARATA) problem. Finally, in Chapter 7, we provide a summary and some future directions for research.

2. BACKGROUND

In this chapter, we provide some background on multi-agent systems, focusing in particular on task allocation and routing/path planning problems. Perhaps, the most well known routing problem is the Traveling Salesperson Problem. We discuss this problem, and provide insight into why it is a hard problem that has attracted much interest from the research community. A short description of the Vehicle Routing Problem (VRP), which generalizes the TSP is then provided, followed by a short discussion of the different objectives commonly seen for these problems. In the last section of this chapter, we provide details of some allied problems, which are also combinatorial in nature.

2.1 Multi-agent Systems

Multi-agent systems consist of agents and the environment they reside in. Such systems, depending on context, may refer to software agents or can also be robots, humans or human teams. A multi-agent system may also contain a combination of humans and robotic agents. In this thesis, we consider robotic agents. In particular, since the focus is on finding routes for agents, the term *agents* refers to mobile robotic agents. For instance, agents can be cars or Unmanned Autonomous Vehicles (UAVs). A taxonomy of task allocation in multi-robot systems is provide by Gerkey and Mataric in [7].

When considering multi-agent systems, the agents in the system may be homogeneous or heterogeneous. The agents are said to be homogeneous if all agents involved are identical, and the agents are said to be heterogeneous otherwise. We can further classify heterogeneous agents based on how the agents differ from each other. The agents are *structurally heterogeneous* if they have different properties such as maxi-

mum speed, turn radius, etc., but they perform the same functions. The agents are called *functionally heterogeneous* if the agents inherently perform different functions. For example, depending on the sensors mounted on a UAV, the UAV has different capabilities and caters to different requirements.

2.2 Traveling Salesperson Problem (TSP)

The Traveling Salesperson Problem (TSP) is one of the most widely studied problem in task allocation and vehicle routing. The problem considers a salesperson who must start from a city, visit N cities and return to his/her start location. The salesperson may visit the cities in any order, so long as all the cities are visited. TSP asks the following question: "Given a list of N cities and the distances between each pairs of cities, what is the shortest route that the salesperson can take to visit all N cities and return to the origin city?"

This problem does not scale well as the number of cities increases. There are $N!$ permutations in which the salesperson can visit the N cities. For $N = 3$, that corresponds to 6 permutations, but for $N = 15$, it is over a trillion permutations. This is therefore a hard problem and in the theory of computational complexity, belongs to a class of problems known as NP-hard problems (discussed in Section 2.5).

There are several variants of TSP in the literature. The *metric TSP*, where the distances are assumed to satisfy the triangle inequality (discussed in Section 2.6), is among the most well studied variants. Christofides' algorithm [8] is a well known algorithm to find approximate solutions to TSP. In particular, this algorithm guarantees to find solutions that have a tour length of at most $\frac{3}{2}$ times the optimal tour length. This algorithm finds a Minimum Spanning Tree (MST) [9,10], and then finds a minimum weight perfect matching on the set of vertices with odd degree in the MST. The union of the MST and the minimum weight perfect matching then yield a tour on the set of cities to be visited. The cities are visited along this tour by shortcutting (or skipping) cities that have already been visited.

Several variants of TSP has been studied in the literature. One popular extension is the case where multiple salespeople are present. An overview and the various applications of the Multiple Traveling Salesperson Problem is provided in [11]. Another variant is where instead of finding a tour for the salesperson (where the salesperson ends his tour at the start location), a path is found. This means that the salesperson is no longer constrained to end his route at the same location as he started. This is known as the Traveling Salesperson Path Problem and is known to be an NP-hard problem [12]. Another class of TSP problems looks at cases with multiple salepersons, but aims to find solutions that minimize the time to tour the set of cities. This is captured by problems that look to find cycle (or tree or path) covers [13]. These can also be thought of as special cases of Vehicle Routing Problems.

The Traveling Purchaser Problem (TPP) and the Vehicle Routing Problem (VRP) are both generalizations of TSP. The traveling purchaser problem (TPP) considers the following: "Given a list of marketplaces, the cost of travelling between different marketplaces, and a list of available goods along with the price of each such good at each marketplace, find the route with the minimum combined cost of purchases and traveling, for a given list of articles." This formulation gives a "weight" or value to each of the marketplaces (analogous to the cities in TSP) depending on the list of goods available at the place and their prices. Another key difference from TSP is that not all marketplaces needs to be visited. This problem can be reduced to TSP by setting the available goods at each market to be distinct (and non-overlapping) and making the list of articles to be purchased such that it contains at least one item from each market. Then, all the marketplaces have to be visited, and thus, the optimal solution will be obtained by optimizing the TSP. A survey on TPP and its variants is provided in [14].

Vehicle Routing Problem (VRP) and its Variants

Vehicle Routing Problem [15], being a generalization of the Traveling Salesperson Problem, belongs to the class of NP-hard problems. In simple terms, the problem considers a set of vehicles and aims to optimally route these vehicles to service a given set of customers.

Some of the common variants of VRP seen in literature are highlighted below:

- Vehicle Routing Problem with Multiple Trips (VRPMT), where the vehicles are allowed to do more than one trip (i.e., they can visit the depot multiple times).
- Open Vehicle Routing Problem (OVRP), where vehicles are not constrained to return to the depot.
- Vehicle Routing Problem with Pickup and Delivery (VRPPD), where commodities need to be picked up from a given set of locations and dropped off at specified locations.
- Vehicle Routing Problem with Time Windows (VRPTW), where the deliveries to each location must be made within certain specified time windows.
- Capacitated Vehicle Routing Problem (CVRP), where vehicles can carry no more than a specified quantity of commodities at a given time (i.e., a capacity constraint is imposed).

2.3 Objectives of Routing Problems

Multi-agent optimization problems may have a variety of objectives. For instance, the objective could be optimizing the quality of a process or method; it could also be optimizing the cost to realize a project. In the context of routing problems, the most common objective considered is minimizing the distance travelled. If we correlate distance travelled to the amount of fuel consumed, then this objective translates to minimizing the fuel consumption and thus the "cost" of the route. Such objectives are

referred to as *min-sum* objectives. Another approach is to minimize the time taken to complete the tasks. If there are multiple agents involved, one could minimize the maximum time taken by any agent to complete its mission. Such objectives are referred to as *min-max* objectives (some works in the literature also refer to it as minimax). In particular, if the application is time critical, then this metric is extremely important (for example, in rescue operations). This is also very relevant to applications where the quality of service is determined by the latency involved.

2.4 Related Problems in the Literature

Job scheduling Problem (JSP) is an optimization problem which aims to assign jobs to available machines in order to optimize the latest completion time of all the jobs. [16]. Suppose there is a set of jobs (of potentially varying processing times) that need to be executed by a set of available machines (with potentially different processing power). JSP tries to allocate jobs to machines so that the jobs are completed by the machines with minimum latency. This objective of minimizing the latency in completion closely aligns with the min-max objective (if the parameter of the min-max cost function is time).

Another closely related problem in the literature is the Orienteering Problem (OP) [17, 18] (which is based on the sport Orienteering) where the start and end points for the path are given along with other locations that have associated scores. The objective is to maximize the score (by visiting associated locations), subject to constraints on the start location, end location and travel time.

2.5 Complexity Theory

We provide a brief background on some of the aspects of complexity theory that we use in this thesis [12]. *Decision problems* are the class of problems which can be posed as a "yes" or "no" question for a given a set of inputs. The complexity class P is defined as the set of decision problems whose answer ("yes" or "no") can be found

in polynomial time (i.e., the time taken to solve these problems scale polynomially with the size of the input). The class NP consists of problems whose answer "yes" is verifiable in polynomial time. NP-hard problems are a class of problems that are "at least as hard as the hardest problems in NP". More formally, a problem $\mathcal{A}$ is NP-hard when every problem $\mathcal{B}$ in NP can be reduced in polynomial time to $\mathcal{A}$.

Optimization problems are a class of problems where we aim to find the "best" solution from the set of all feasible solutions. Since these problems do not have a "yes" or "no" answer, by definition, they do not belong to the class NP. However, they can still be NP-hard, if there is a polynomial time reduction from an NP-hard decision problem to the given optimization problem. An approximation algorithm for an optimization problem is an algorithm that efficiently finds approximate solutions to the problem with provable guarantees on the performance of the solution with respect to the optimal solution. In particular, an α-approximation algorithm finds an approximate solution (in time that scales polynomially with the size of the input) such that the approximate solution is no worse than a factor α of the optimal solution.

2.6 Assumptions

We consider task allocation and routing of agents in this thesis. Agents typically travel in a Euclidean space. Therefore, a natural assumption to make is that the triangle inequality holds. The triangle inequality states that for any triangle, the sum of the lengths of any two sides must be greater than or equal to the length of the remaining side. In our context, this implies that the direct distance between any two points must be no longer than the distance to reach it through other points.

We consider problems with finite representations in this thesis. This means that the values can be scaled so that they are integers. Thus, all distances and weights considered in this thesis are taken to be non-negative integers.

3. MIN-MAX TOURS FOR TASK ALLOCATION TO HETEROGENEOUS AGENTS

Multi-robot systems will play a large role in a variety of modern and future applications including exploration, surveillance, search and rescue operations, cooperative control, and operations in hazardous environments. In order to effectively utilize these multi-robot systems, it is necessary to allocate an appropriate set of tasks to each robot or agent in the system. Such problems have been widely considered in the literature [4, 5, 19–21], most typically for the case where all agents are the same. However, future multi-robot systems are also projected to have a large amount of diversity in terms of the capabilities of the agents, and the applications will consist of tasks that can only be done by agents that possess certain capabilities [22–25]. We address this problem in this chapter, namely allocating tasks efficiently to heterogeneous agents while meeting task-agent compatibility constraints.

An overview of the work on the Traveling Salesperson Problem (TSP) with multiple (homogeneous) salespersons, where the sum of the cost of tours by all salespersons is minimized, is given in [11]. Variants of TSP with multiple depots are considered in [26] and [27]. In [28] and [29], the case where two heterogeneous vehicles (with associated travel costs) start from distinct initial locations and jointly visit a set of targets is studied. As opposed to the above works that minimize the sum of the costs of the tours, the following works focus on minimizing the maximum tour cost incurred by any agent in a group of homogeneous agents. A tour splitting heuristic for the k-person variant of TSP that minimizes the cost of the largest tour is given in [30], while [31] considers the case with multiple depots. Approximation algorithms for a problem known as the min-max tree cover problem are provided in [13, 32, 33]. While the above works consider homogeneous salespersons (or robots/agents), the recent work by [5] gives a decentralized auction-based task allocation for heterogeneous

robots (with different constraints on their motion) to minimize the total time taken to perform all tasks. In [22], a swarm of heterogeneous robots (of different types, with each type having different traits) is required to be distributed among a set of tasks that require specialized capabilities. They optimize the transition rates for each type of robot so that the desired trait distribution is reached, but do not consider travel time between tasks. The works [34] and [35] consider task allocation to heterogeneous agents but do not consider tours for agents to traverse as we do.

In this chapter, we combine the idea of heterogeneity in agent functionality with that of minimizing the maximum cost incurred by any agent to tour tasks. Specifically, consider a scenario where a set of tasks at different locations need to be executed; however, not all tasks can be done by all agents, and certain task-agent compatibility constraints must be satisfied. Agents are partitioned into different types based on the capabilities of the agents. Tasks are partitioned into sets of type-specific tasks and generic tasks, where type-specific tasks can only be performed by agents of a given type and generic tasks can be performed by any agent. To capture this scenario, we present the Heterogeneous Agent Cycle Problem (HACP) which aims to allocate a set of tasks among heterogeneous agents such that the maximum time to tour the tasks by any agent is minimized. This is an important metric, especially when the tasks are time critical or when the quality of service is characterized by maximum delay. Table 3.1 gives a concise overview of the literature in related multi-agent path planning problems. The recent work by [36] considers the vehicle routing problem with compatibility constrains, where certain tasks are constrained to be executed only by certain agents. They provide an algorithm with an approximation factor of $2\lceil \ln n \rceil + 1$. While their notion of constraints aligns with our notion of functional heterogeneity, our algorithm has a constant approximation factor.

The rest of this chapter is organized as follows. We start with our problem formulation. We then present a naive algorithm and show that the approximation factor of this algorithm increases linearly with the number of agents. This motivates the need to develop better algorithms that perform well as the number of agents increase. We

Table 3.1.
Overview of work on related multi-agent path planning problems

Objective	Heterogeneity	Number of depots	Work	Description
Min-sum	None	One, multiple	[11]	Survey of various formulations of multiple traveling salesman problems.
Min-sum	Structurally heterogeneous	Two	[29]	Primal-dual 2-approximation algorithm provided.
Min-sum	Structurally heterogeneous	Multiple	[19, 37]	[19] provides an exact algorithm using a branch and cut algorithm. [37] provides a 2-approximation algorithm based on the primal-dual method.
Min-max	None	Multiple	[38]	Provides a heuristic based on region partitioning, and a linear programming-based approach.
Min-max	Structurally heterogeneous	Two	[6]	Heuristics based on primal-dual technique.
Min-max	Structurally heterogeneous	Multiple	[39]	Distributed algorithm based on gossip communication.
Min-max	**Functionally heterogeneous**	**One**	**[36, 40]**	**Our work (published in [40]) provides a 5-approximation algorithm and is discussed in this chapter.**
Min-max	**Functionally heterogeneous**	**Multiple**	**[36, 41]**	**Our work (published in [41]) is discussed in Chapter 4.**

present two such algorithms and show that these are 5-approximation algorithms for HACP. We also show that in the special case where each of the heterogeneous agents are distinct, the proposed algorithms haves an approximation factor of 4. The results presented in this chapter were published in [40].

3.1 Problem Statement

Consider a set of tasks T that are to be completed by a set of k heterogeneous agents $A = \{A_1,\ A_2,\ldots,\ A_k\}$. Each agent is one of m *types*. Let $f : \{1,\ldots,k\} \rightarrow \{1,\ldots,m\}$ be a function that takes an agent number as input and outputs the type of that agent. For each $i \in \{1,2,\ldots,m\}$, let m_i be the number of agents of type i, with $\sum_{i=1}^{m} m_i = k$. Let T be composed of two broad classes of tasks: type-specific tasks and generic tasks. Type-specific tasks can be performed only by a specific type of agent, whereas generic tasks can be performed by any agent. Let T_0 denote the set of generic tasks and $T_i,\ 1 \leq i \leq m$, denote the set of type-specific tasks that can be performed by agents of type i. Thus, $T = T_0\ \cup\ (\cup_{i=1}^{m} T_i)$. Let all agents start at the start node v_s.

Consider a complete graph $G = (V, E)$ with vertex set $V = T \cup \{v_s\}$ and edge set $E = \{(u, v) : u, v \in V,\ u \neq v\}$. Let each edge $e = (u, v) \in E$ have weight $d(u, v)$ given by the distance between the nodes u and v. Let the direct travel cost between two nodes be the weight of the edge connecting the two nodes in G. We assume the distances satisfy the triangle inequality.[1] The cost of executing a task is assumed to be very small compared to travel costs and is hence neglected. A tour on a set of nodes $V' \subseteq V$ is a closed path from the start node v_s through all nodes in V' ending at v_s. The cost of the tour is defined as the sum of weights of all edges on that tour. Let $C^*(V')$ denote the tour cost of an optimal tour on the set V'. The objective of the allocation problem is to partition the set of tasks T among the agents subject

[1]All distances and weights will be taken to be non-negative integers as we are interested in problems that have finite representations.

to the task-agent compatibility constraints, such that the maximum cost among all agents to tour their allocated tasks is minimized. This is framed as follows.

Heterogeneous Agent Cycle Problem (HACP):

$$\begin{aligned} \min_{S_1,S_2,\ldots,S_k \subseteq T} \ & \max_{1 \le j \le k} \ C^*(S_j) \\ \text{subject to} \quad & \cup_{j=1}^{k} S_j = T, \ S_j \cap S_i = \emptyset, \ \forall j \neq i, \\ & S_j = V_j \cup R_j, \ \forall j \in \{1, 2, \ldots, k\}, \\ & V_j \subseteq T_{f(j)}, \ R_j \subseteq T_0, \end{aligned}$$

where S_j is the task set allocated to agent A_j, $1 \le j \le k$. The first constraint implies that each task must be executed by exactly one agent. The remaining conditions state that the task set allocated to each agent A_j is a union of type-specific tasks V_j (which is a subset of $T_{f(j)}$, where $f(j)$ is the type of agent A_j) and generic tasks R_j (which is a subset of T_0).

Any instance of the Traveling Salesperson Problem (TSP) can be trivially reduced to an instance of HACP by setting the number of agents $k = 1$ and all tasks to be generic. Thus, the HACP is trivially NP-Hard, and has no polynomial time solution unless P= NP. Hence, in the rest of this chapter we develop approximation algorithms for HACP.

3.2 A Naive Algorithm for HACP

We start with the following simple algorithm to solve HACP. In this algorithm, we first select one agent of each type. To each of these selected agents, we allocate all the type-specific tasks associated with that agent type. All remaining tasks (i.e., generic tasks) are then allocated to agent A_1. Each agent now computes a tour using some approximation algorithm (e.g., Christofides' algorithm [8]) on the set of tasks allocated to it. Algorithm 1 describes this naive allocation.

Algorithm 1 NaiveAllocation Algorithm

1: **procedure** NaiveAllocation(A, T, G)

2: For each agent type $i \in \{1, \ldots, m\}$, select one agent from agents of type i. Allocate type-specific tasks T_i to this agent.

3: Allocate generic tasks T_0 to agent A_1.

4: Compute tour (starting and ending at node v_s) for the set of tasks allocated to each agent using an approximation algorithm.

5: Return tours for each agent.

6: **end procedure**

Theorem 3.2.1 *Suppose the algorithm that is used to compute the tour for each agent in line 4 of the* NaiveAllocation *algorithm has approximation factor α. Then,* NaiveAllocation *is an αk-approximation algorithm for HACP.*

Proof For $1 \leq j \leq k$, let S_j^* be the set of tasks allocated to agent A_j under the optimal allocation for HACP. Note that $S_j^* \subseteq T \ \forall j$ and $\cup_{j=1}^{k} S_j^* = T$. The set $S_j^* \subseteq T_{f(j)} \cup R_j^*$ where R_j^* is the subset of tasks in T_0 allocated to agent A_j under the optimal allocation policy.

For $1 \leq j \leq k$, let S_j denote the set of tasks allocated to agent A_j by NaiveAllocation. Let $C^*(\cdot)$ denote the cost to optimally tour a given set of tasks (starting and ending at v_s) and let $C_{NA}(\cdot)$ denote the tour costs returned by the NaiveAllocation algorithm. The approximation ratio R for the NaiveAllocation algorithm is given by

$$R = \frac{\max\limits_{1\leq j\leq k} C_{NA}(S_j)}{\max\limits_{1\leq j\leq k} C^*(S_j^*)} \leq \frac{C_{NA}(T)}{\max\limits_{1\leq j\leq k} C^*(S_j^*)}$$
$$\leq \frac{\alpha C^*(T)}{\max\limits_{1\leq j\leq k} C^*(S_j^*)} \leq \frac{\alpha(\sum_{j=1}^{k} C^*(S_j^*))}{\max\limits_{1\leq j\leq k} C^*(S_j^*)} \leq \alpha k.$$

The inequality $C_{NA}(T) \leq \alpha C^*(T)$ holds as the tour in line 4 of NaiveAllocation is computed using an α-approximation algorithm. Also, $C^*(T) \leq \sum_{j=1}^{k} C^*(S_j^*)$ follows since $S_j^*,\ 1 \leq j \leq k$, form a partition of T and the triangle inequality holds. ■

The previous theorem shows that the approximation factor of even a naive algorithm as the one described above is bounded, but the bound grows linearly with the number of agents k. This motivates us to look for better algorithms for HACP, in particular, algorithms that perform well as the number of agents become large. We provide two constant factor algorithms in the following section.

3.3 Constant Factor Approximation Algorithms for HACP

3.3.1 Cycle Splitting Approach

Consider an instance of HACP. In order to find an allocation of tasks to agents, we must allocate type-specific tasks among agents of the required type and allocate generic tasks among all agents. We approach the problem by handling these two allocations separately.

Given a set of tasks T_i and a number k, let *TaskSplitter* be any algorithm that splits the set of tasks T_i into k sub-tours $\{T_{i1}, T_{i2}, ..., T_{ik}\}$ within some factor β of the optimal split (in the min-max sense). For example, Frederickson et. al. [30] give a tour splitting heuristic that takes a tour T' on a set of locations to be visited (first node of which is set as the start node), and a positive integer k as input and gives a set of k subtours (starting and ending at the start node) as the output. The cost of these subtours are within a factor of $1 + F - 1/k$ of the optimal min-max cost, where F is the approximation factor of the algorithm used to generate the initial tour T'. We provide Algorithm 2 to allocate a given set of tasks T to a group of k heterogeneous agents A.

Algorithm 2 Min-Max Tour by CycleSplit Algorithm

1: **procedure** CycleSplit(A, T, G)
2: **for** each agent type i, $1 \leq i \leq m$ **do**
3: Find Christofides' tour (starting and ending at v_s) on the set of type-specific tasks of type i.
4: Use *TaskSplitter* to get m_i subtours (starting and ending at v_s) on set T_i.
5: Allocate one subtour to each agent of type i.
6: **end for**
7: Find Christofides' tour (starting/ending at v_s) on T_0.
8: Use *TaskSplitter* to split tour on T_0 into k subtours (starting and ending at v_s), denoted by $\{R_1, R_2, \ldots, R_k\}$.
9: Allocate subtour R_j to agent A_j, $1 \leq j \leq k$.
10: Combine the type-specific task subtour and generic task subtour allocated to each agent.
11: Return a tour for each agent.
12: **end procedure**

Theorem 3.3.1 CycleSplit *is a* 2β*-approximation algorithm for HACP, where* β *is the approximation factor of the algorithm TaskSplitter used in steps 4 and 8.*

Proof For $1 \leq j \leq k$, let S_j^* be the allocation of tasks to agent A_j under an optimal algorithm for HACP. Then, S_j^* can be expressed as $S_j^* = V_j^* \cup R_j^*$, where V_j^* is the subset of type-specific tasks allocated to agent A_j and R_j^* is the subset of generic tasks assigned to agent A_j. Let $S_j = V_j \cup R_j$ be the allocation to agent A_j by the CycleSplit algorithm, where V_j is the subset of type-specific tasks allocated to agent A_j and R_j is the subset of generic tasks allocated to agent A_j. Let $C^*(\cdot)$ denote the cost to optimally tour a given set of tasks starting and ending at node v_s. Let $C_{TS}(R_j)$ and $C_{TS}(V_j)$ denote the cost of the subtours on R_j and V_j returned by *TaskSplitter* in steps 4 and 8 respectively. Let $C_{CS}(S_j)$ denote the cost of a tour on

S_j returned by the CycleSplit algorithm. Thus, the approximation factor R of the CycleSplit algorithm is given by

$$\begin{aligned} R = \frac{\max\limits_{1\le j\le k} C_{CS}(S_j)}{\max\limits_{1\le j\le k} C^*(S_j^*)} &\le \frac{\max\limits_{1\le j\le k}\{C_{TS}(V_j)+C_{TS}(R_j)\}}{\max\limits_{1\le j\le k} C^*(S_j^*)} \\ &\le \frac{\max\limits_{1\le j\le k} C_{TS}(V_j)}{\max\limits_{1\le j\le k} C^*(S_j^*)} + \frac{\max\limits_{1\le j\le k} C_{TS}(R_j)}{\max\limits_{1\le j\le k} C^*(S_j^*)} \\ &\le \frac{\max\limits_{1\le j\le k} C_{TS}(V_j)}{\max\limits_{1\le j\le k} C^*(V_j^*)} + \frac{\max\limits_{1\le j\le k} C_{TS}(R_j)}{\max\limits_{1\le j\le k} C^*(R_j^*)}, \end{aligned} \tag{3.1}$$

where we use the facts that $S_j = V_j \cup R_j$, $S_j^* = V_j^* \cup R_j^*$ and that the triangle inequality holds.

Consider a case with the same set of type-specific tasks as above, but with no generic tasks. For this case, let $V_1', V_2', \ldots, V_k'$ denote the set of tasks allocated to agents under an optimal allocation (in the min-max sense). Since the CycleSplit algorithm allocates type-specific tasks independent of generic tasks, the allocation under this algorithm will be $V_1, V_2, \ldots, V_k$. Thus,

$$\max_{1\le j\le k} C_{TS}(V_j) \le \beta \max_{1\le j\le k} C^*(V_j'). \tag{3.2}$$

The inequality above follows from the fact that *TaskSplitter* algorithm splits a tour into subtours that are within a factor β of the optimal min-max cost. Next, note that the min-max cost to optimally tour $\{V_j'\}$ cannot exceed the min-max cost to optimally tour $\{V_j^*\}$, by the optimality of the partition $\{V_j'\}$, $1 \le j \le k$. Thus, from (3.2),

$$\max_{1\le j\le k} C_{TS}(V_j) \le \beta \max_{1\le j\le k} C^*(V_j') \le \beta \max_{1\le j\le k} C^*(V_j^*). \tag{3.3}$$

Using a similar procedure as before, this time comparing the allocation of a case with no type-specific tasks against the set of all tasks, we get,

$$\max_{1\le j\le k} C_{TS}(R_j) \le \beta \max_{1\le j\le k} C^*(R_j^*). \tag{3.4}$$

From equations (3.1), (3.3) and (3.4), we get $R \le 2\beta$. ■

We now refine the above bound using the $(1 + F - \frac{1}{k})$-approximation algorithm *SPLITOUR*[2] from [30] that splits a tour (starting and ending at a start-node) on a set of nodes into k subtours each starting and ending at the start node, where F is the approximation factor of the algorithm used for constructing the initial tour.

Corollary 3.3.2 CycleSplit *algorithm using the SPLITOUR algorithm from [30] as TaskSplitter (in lines 4 and 8) is a* $(5 - \frac{2}{k})$*-approximation algorithm for HACP. In the special instance of HACP where there is one agent of each type (i.e., distinct heterogeneous agents),* CycleSplit *has an approximation factor of* $(4 - \frac{1}{k})$.

Proof The algorithm *SPLITOUR* is a $\frac{5}{2} - \frac{1}{k}$ factor algorithm to split a Christofides' tour into k subtours (since Christofides provides an $F = \frac{3}{2}$ factor approximation for the initial tour). Equation (3.1) can be written as

$$R \leq \left(\frac{5}{2} - \frac{1}{\max\limits_{1 \leq i \leq m} m_i}\right) + \left(\frac{5}{2} - \frac{1}{k}\right) \leq 5 - \frac{2}{k}, \tag{3.5}$$

where, the last inequality is obtained by upper bounding $\max\limits_{1 \leq i \leq m} m_i$ with k. In the special case with distinct heterogeneous agents, $m_i = 1,\ \forall i$. Thus $\max\limits_{1 \leq i \leq m} m_i = 1$. The result follows from substituting these values in equation (3.5). ■

Note that the CycleSplit algorithm, when using *SPLITOUR* as *TaskSplitter* in lines 4 and 8, splits a TSP tour evenly into subtours and allocates one subtour to each agent. This approach, when splitting the generic tasks, may not perform well when agents have different allocations of type-specific tasks. To address this, we propose a modified algorithm in the following section.

3.3.2 Min-Max Splitting Approach for Heterogeneous Agents

Based on the intuition gained from the CycleSplit algorithm, we propose a modified algorithm called HeteroMinMaxCycleSplit. Instead of splitting a tour

[2]Frederickson, Hecht and Kim call the algorithm *k-SPLITOUR* in their paper; however, we refer to it as *SPLITOUR* in order to avoid confusing k with the number of agents as we use it in this thesis.

on the set of generic tasks into nearly equal segments, this algorithm allocates the generic tasks to agents based on the cost incurred by agents to tour its type-specific tasks. In the HeteroMinMaxCycleSplit algorithm, we allocate tasks to agents in three phases:

- **Phase 1**: Type-specific task allocation
- **Phase 2**: Generic task allocation (accounting for Phase 1 allocation)
- **Phase 3**: Rebalancing tasks within agents of each type.

In Phase 1, type-specific tasks are allocated among agents of the associated type as in CycleSplit. The generic tasks are allocated among all agents in Phase 2. Unlike in CycleSplit, the allocation during Phase 2 tries to balance the total cost incurred by agents by taking into account the allocation to agents after Phase 1. After Phase 2, all tasks allocated to agents of the same type can be done by all agents of that type. Thus, in Phase 3, we try to re-balance the load among agents of type $i, 1 \leq i \leq m$. Tasks allocated to agents of type i are pooled into a set T_i' and re-allocated among the agents of type i by splitting the Christofides' tour on the set T_i' (starting and ending at v_s) into m_i sub-tours using *SPLITOUR* [30]. If min-max cost of this set of subtours is lesser than that at the end of Phase 2 (for agents of type i), then this allocation is adopted; otherwise, the allocation at the end of Phase 2 is retained.

We formally present the algorithm in two steps. We first give an algorithm HeteroCycleSplit that takes an additional input: a positive integer λ denoting the desired upper bound for the tour length for any agent. This algorithm finds tours (if they exist) for each agent such that the cost for each agent's tour is less than λ. The value of λ cannot be less than than twice the largest edge from v_s to any task location, i.e., $2 \max_{t \in T} d(v_s, t)$, where $d(v_s, t)$ is the distance from v_s to task t. Furthermore, let $C(\cdot)$ be the cost of Christofides' tour on the given set of tasks. Then, λ cannot exceed $\max_{1 \leq i \leq k} C(T_i) + C(T_0)$. The HeteroMinMaxCycleSplit algorithm performs a binary search on λ, running HeteroCycleSplit in each iteration, to find the best set of tours for the agents.

We now provide a brief sketch of the algorithm *SPLITOUR* from [30] for the purpose of applying it as the *TaskSplitter* algorithm. It takes as input a set of n nodes $v_1, \ldots, v_n$ (with the initial node as the start node) and a positive number k.

Algorithm 3 Heterogeneous Task Split within bound λ

procedure HETEROCYCLESPLIT(A, T, G, λ)

▷ **Phase 1:** Type-specific task allocation

for each agent type i, $1 \leq i \leq m$ **do**

Find Christofides' tour (starting and ending at v_s) on the set of type-specific tasks of type i.

Use *TaskSplitter* to split the tour on T_i into m_i subtours (starting and ending at v_s).

Allocate one subtour to each agent of type i.

end for

▷ **Phase 2:** Generic task allocation

Mark all agents as free. Remove all vertices in $\cup_{i=1}^{m} T_i$ from G and all edges incident on these vertices. Denote the resulting graph by G'.

Mark tasks in G' as unallocated. Find Christofides' tour H (starting and ending at v_s) on nodes in G'.

Consider the next unallocated task, say t, along H starting from v_s. For each free agent A_j, find cost to tour (starting and ending at v_s) all tasks allocated to it along with t appended at the end of its tour. If no free agent can add the task to its tour without exceeding cost λ, return failure. Else, select the agent with the minimum cost.

Allocate t to the selected free agent. Keep allocating unallocated tasks along H to this agent (by appending tasks at the end of its tour successively) as long as the agent's tour cost does not exceed λ.

Remove the tasks allocated in the previous step from the set of unallocated tasks and mark agent as busy.

Go to step 9 if the set of unallocated tasks is non-empty.

▷ **Phase 3:** Rebalancing within agent types

for each agent type i, $1 \leq i \leq m$ **do**

Let $\mathcal{A}_i$ be the set of tours allocated to agents of type i after Phase 2.

Deallocate tasks from all agents of type i into a set T_i'. Find Christofides' tour H' on $T_i' \cup \{v_s\}$ (starting and ending at v_s).

Run *TaskSplitter* on H' to get one subtour for each agent of type i. Denote this set of tours (subtours of H') by $\mathcal{A}_i'$.

If min-max cost of the tours in $\mathcal{A}_i'$ is less than the min-max cost of the tours in $\mathcal{A}_i$, allocate the tours in $\mathcal{A}_i'$ to agents of type i (a different tour to each agent); else, allocate the tours in $\mathcal{A}_i$ to agents of type i (a different tour to each agent).

end for

Return tours for each agent.

end procedure

Algorithm 4 Min-Max Tour by HeteroMinMaxCycleSplit Algorithm

procedure HeteroMinMaxCycleSplit(A, T, G)

Do binary search in the interval $[2 \max_{t \in T} d(v_s, t), \ \max_{1 \leq i \leq k} C(T_i) + C(T_0)]$ to find the smallest value of λ for which HeteroCycleSplit(A, T, G, λ) returns a set of valid tours.

Return the set of tours returned by HeteroCycleSplit(A, T, G, λ).

end procedure

First, the algorithm constructs a Christofides' tour on all the nodes. Let L be the cost of this tour and let c_{max} be the maximum direct distance of any node from the start node, i.e., $c_{max} = \max_{1<i\leq n} d(v_1, v_i)$. For $1 \leq j < k$, it finds the largest vertex $v_{p(j)}$ along the tour such that the cost to traverse the tour from the start node to $v_{p(j)}$ does not exceed $\frac{j}{k}(L - 2c_{max}) + c_{max}$. It returns k subtours $(v_1, \ldots, v_{p(1)}, v_1)$, $(v_1, v_{p(1)+1}, \ldots, v_{p(2)}, v_1)$, …, $(v_1, v_{p(k-1)+1}, \ldots, v_n, v_1)$, each of which has cost less than $\frac{1}{k}(L - 2c_{max}) + 2c_{max}$. Frederickson et. al. [30] show that if the length of the subtours

do not exceed $\frac{1}{k}(L - 2c_{max}) + 2c_{max}$, then the min-max cost of the subtours is no worse than a factor $(\frac{5}{2} - \frac{1}{k})$ of the optimal min-max cost.

The following theorem uses the splitting heuristic summarized above to bound the performance of our algorithm.

Theorem 3.3.3 HETEROMINMAXCYCLESPLIT *is a* $(5 - \frac{2}{k})$*-approximation algorithm for HACP with k heterogeneous agents when SPLITOUR is used as TaskSplitter in lines 4 and 16 of* HETEROCYCLESPLIT.

Proof Let λ_1 be the maximum cost among all the subtours of type-specific tasks after line 6 (Phase 1) of the HETEROCYCLESPLIT algorithm, i.e., $\lambda_1 = \max_{1 \leq j \leq k} C_{TS}(V_j)$, where V_j denotes the set of type-specific tasks allocated to agent A_j and $C_{TS}(\cdot)$ denotes the cost associated with the subtour returned by *SPLITOUR*.

Running *SPLITOUR* on a Christofides' tour on the set of generic tasks T_0 (and v_s) to split it into k subtours will return subtours of cost no more than $\frac{1}{k}(L-2c_{max})+2c_{max}$, where L is the length (or cost) of the initial Christofides' tour on $T_0 \cup \{v_s\}$, and c_{max} is the maximum direct distance of any node in T_0 from the start node. No matter how each of these subtours are matched to agents (one subtour per agent), the total min-max cost is no more than $\lambda_1 + \frac{1}{k}(L - 2c_{max}) + 2c_{max}$.

Set λ to be $\lambda_1 + \frac{1}{k}(L - 2c_{max}) + 2c_{max}$. Consider HETEROCYCLESPLIT with this value of λ. In Phase 1, HETEROCYCLESPLIT allocates type-specific tasks to agents, same as steps 2-6 of CYCLESPLIT. The maximum cost of any agent's tour after Phase 1 is λ_1. In Phase 2, HETEROCYCLESPLIT computes a Christofides' tour H on the set of generic tasks and v_s. The algorithm selects an agent that has minimum cost to complete its current allocated tasks in addition to the next task along H. The algorithm then allocates tasks along the tour H to the selected agent as long as the cost does not exceed λ. Regardless of which agent gets selected first, the set of tasks that are allocated to the selected agent in HETEROCYCLESPLIT will contain the tasks in the first subtour generated by *SPLITOUR*. The allocation by HETEROCYCLESPLIT to the first selected agent may contain more tasks than the

first subtour of *SPLITOUR*, but not less. Thus, after allocating tasks to the first agent, the number of tasks left to be allocated to the remaining $k-1$ agents in HeteroCycleSplit is no more than the number of tasks left in the $k-1$ subtours to be allocated to the remaining $k-1$ agents in *SPLITOUR*.

Given that the set of tasks left for allocation in HeteroCycleSplit is a subset of the set of tasks left to be allocated in *SPLITOUR*, they can be grouped into subtours that each have cost at most $\frac{1}{k}(L-2c_{max})+2c_{max}$. Thus, it is guaranteed that the remaining tasks can be allocated to other $k-1$ agents in HeteroCycleSplit algorithm for the specified value of λ. Thus, by inducting on the number of agents to which tasks have been allocated along the initial tour, the HeteroCycleSplit algorithm is guaranteed to return a feasible solution to HACP for $\lambda = \lambda_1 + \frac{1}{k}(L-2c_{max})+2c_{max}$. Since HeteroCycleSplit checks tour cost against λ before allocation of tasks to an agent, it guarantees that the tour costs for all agents is no more than λ. The approximation factor R is given by

$$
\begin{aligned}
R &\leq \frac{\lambda}{\max\limits_{1\leq j\leq k} C^*(S_j^*)} \\
&= \frac{\lambda_1}{\max\limits_{1\leq j\leq k} C^*(S_j^*)} + \frac{\frac{1}{k}(L-2c_{max})+2c_{max}}{\max\limits_{1\leq j\leq k} C^*(S_j^*)} \\
&\leq \frac{\max\limits_{1\leq j\leq k} C_{TS}(V_j)}{\max\limits_{1\leq j\leq k} C^*(V_j^*)} + \frac{\frac{1}{k}(L-2c_{max})+2c_{max}}{\max\limits_{1\leq j\leq k} C^*(R_j^*)} \qquad (3.6) \\
&\leq \left(\frac{5}{2} - \frac{1}{\max\limits_{1\leq i\leq m} m_i}\right) + \left(\frac{5}{2} - \frac{1}{k}\right) \leq 5 - \frac{2}{k}, \qquad (3.7)
\end{aligned}
$$

where equation (3.6) is given by the triangle inequality and the fact that V_j^* and R_j^* are subsets of S_j^* (the optimal allocation to agent A_j). Equation (3.7) is obtained from equation (3.3) and [30] (summarized prior to this theorem). Note that Phase 3 does not increase the bound on the approximation factor.

Since all tour costs are integers, given that HeteroCycleSplit returns a feasible solution to HACP for $\lambda = \lambda_1 + \frac{1}{k}(L-2c_{max})+2c_{max}$, it is also guaranteed to return a feasible solution to HACP for $\lambda = \lambda_1 + \lfloor \frac{1}{k}(L-2c_{max})+2c_{max} \rfloor$. Het-

eroMinMaxCycleSplit performs a binary search over λ and is guaranteed to find the above λ for use by HeteroCycleSplit. Thus, HeteroMinMaxCycleSplit is a $(5 - \frac{2}{k})$-approximation algorithm for HACP. ■

Corollary 3.3.4 HeteroMinMaxCycleSplit *is a* $(4 - \frac{1}{k})$*-approximation algorithm for the special instance of HACP where all agents are distinct.*

Proof In this instance, $m_i = 1$, for $1 \leq i \leq m$. The result follows from substituting $m_i = 1$ in equation (3.7). ■

From Theorem 3.3.3, we see that HeteroMinMaxCycleSplit has an approximation factor of 5 regardless of the value of k. In the special instance where all agents are distinct, we see from Corollary 3.3.4 that HeteroMinMaxCycleSplit is a 4-approximation algorithm regardless of the value of k.

Phase 1 of HeteroCycleSplit has a time complexity $\mathcal{O}\left(\sum_{i=1}^{m} \rho(G_i, m_i)\right)$, where $\mathcal{O}\left(\rho(G_i, m_i)\right)$ is the complexity of *TaskSplitter* to return m_i tours on a graph G_i. Phase 2 has a complexity of $\mathcal{O}(n_0^3)$, where n_0 is the number of generic tasks (associated with finding the Christofides' tour in step 8 [42]). Phase 3 has a complexity of $\mathcal{O}(\rho(G_i \cup G', m_i))$, for each of the m agent types. Thus, the complexity of HeteroMinMaxCycleSplit is given by $\mathcal{O}\left(\left(\sum_{i=1}^{m} \rho(G_i, m_i) + n_0^3 + \sum_{i=1}^{m} \rho(G_i \cup G', m_i)\right) \log \sum_{e \in \mathsf{E}(G)} w(e)\right)$, where the log term is due to the binary search over λ, and $w(e)$ is the weight of edge e. When using *SPLITOUR* with Christofides' algorithm as *TaskSplitter*, $\rho(G, m) = n^3$ [42], where n is the number of nodes in G. This results in an overall complexity of $\mathcal{O}(mn^3 \log \sum_{e \in \mathsf{E}(G)} w(e))$ for HeteroMinMaxCycleSplit.

3.3.3 Examples Illustrating the Algorithms

The HeteroMinMaxCycleSplit algorithm considers the allocation of type-specific tasks while allocating generic tasks, allowing it to outperform CycleSplit

when the type-specific tasks are not uniformly distributed among agents. We illustrate this through Example 3.3.5. Example 3.3.6 illustrates the benefit of Phase 3, and shows how the reallocation in Phase 3 can improve performance.

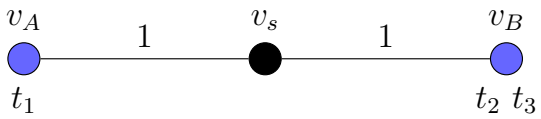


Fig. 3.1. Task locations in Example 3.3.5

Example 3.3.5 *Consider two agents: A_1 of type 1 and A_2 of type 2, located at v_s. Task t_1 is of type 1 and tasks t_2, t_3 are generic tasks, i.e., $T_1 = \{t_1\}$, $T_2 = \emptyset$, $T_0 = \{t_2, t_3\}$. Task t_1 is located at node v_A and tasks t_2 and t_3 are located at node v_B as shown in Figure 3.1.*

CycleSplit *allocates type-specific task t_1 to agent A_1 and splits generic tasks $\{t_2, t_3\}$ among agents A_1 and A_2. Hence, A_1 is allocated tasks $\{t_1, t_2\}$ and A_2 is allocated $\{t_3\}$. The tour costs for agents A_1 and A_2 are 4 and 2 respectively, Thus, the min-max tour cost is* $\max\{4, 2\} = 4$.

HeteroMinMaxCycleSplit *allocates type-specific task t_1 to agent A_1 in Phase 1. In Phase 2, generic tasks $\{t_2, t_3\}$ are split among agents A_1 and A_2 based on remaining capacity. In particular, for $\lambda = 2$,* HeteroCycleSplit *allocates $\{t_1\}$ to agent A_1 and $\{t_2, t_3\}$ to agent A_2. The min-max tour cost for the tour returned by* HeteroMinMaxCycleSplit *is thus* $\max\{2, 2\} = 2$*, which is also the optimal cost in this case.*

Example 3.3.6 *Consider a scenario with two type 1 agents A_1 and A_2 located v_s. Let type-specific tasks $T_1 = \{t_1, t_2\}$ for type 1 agents be located at v_A, which is at a distance of 1 unit from v_s as shown in Figure 3.2. Generic tasks $T_0 = \{t_3, t_4\}$ are located at v_B at a distance of 1 unit from v_s.*

In Phase 1, one type-specific task gets allocated to each of the agents; say t_1 gets allocated to agent A_1 and t_2 gets allocated to agent A_2. In Phase 2, one generic

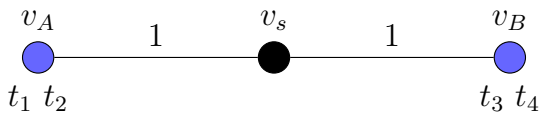


Fig. 3.2. Task locations in Example 3.3.6

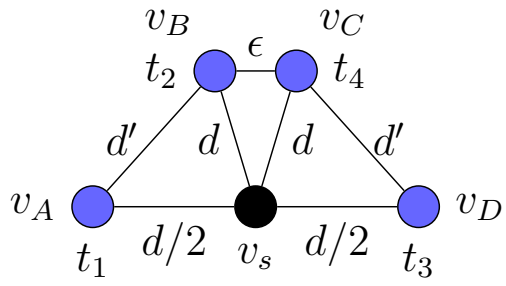
task gets allocated to each of the agents; say t_3 *gets allocated to agent* A_1 *and* t_4 *gets allocated to agent* A_2*. So at the end of Phase 2, both agents need to visit nodes* v_A *and* v_B*.*

In Phase 3, all tasks allocated in the previous phases to agents of the same type are deallocated and redistributed amongst them. In this phase, tasks $\{t_1, t_2, t_3, t_4\}$ *are pooled and SPLITOUR is run on this set. Thus, one agent gets tasks at node* v_A *and the other agent gets tasks at node* v_B*. In this example, Phase 3 reduces the tour cost from* 4 *to* 2 *for both agents, thus reducing the min-max cost by a factor of 2. Note that in this case, the optimal tour cost is also* 2*.*

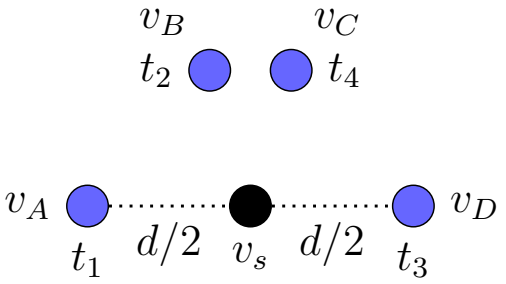
In the following example, we illustrate the importance of step 17 of HeteroCycleSplit (Algorithm 3). We provide an example where pooling the tasks allocated to all agents of a type and reallocating them performs worse than the allocation in Phase 2. In such cases, step 17 ensures that the better allocation is retained.

Example 3.3.7 *Consider a scenario with two type 1 agents* A_1 *and* A_2 *located at* v_s*. Let* $T_1 = \{t_1, t_2\}$ *be the type-specific tasks for type 1 agents and let* $T_0 = \{t_3, t_4\}$ *be the generic tasks. The tasks are located as shown in Figure 3.3(a). Distances between nodes that are not directly connected are defined as the sum of distances along the shortest path between the nodes (the graph can be considered as a road network that the agents can traverse). Let distance between nodes* v_A *and* v_B *be* d' *(which is the same as the distance between nodes* v_C *and* v_D*). Let* $d, d' \gg \epsilon$*. Then, as* $\epsilon \to 0$*,* $d' \to \frac{\sqrt{5}}{2} d$*.*

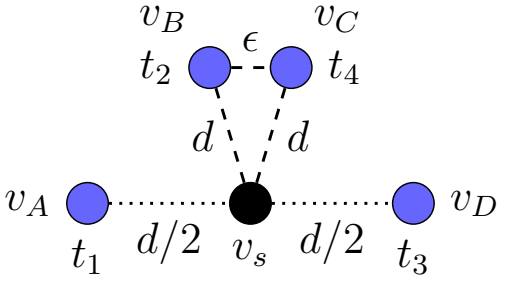
In Phase 1, type-specific task $\{t_1\}$ *is allocated to agent* A_1 *and* $\{t_2\}$ *is allocated to* A_2*. In Phase 2, generic task* $\{t_3\}$ *is allocated to* A_1 *and* $\{t_4\}$ *is allocated to* A_2*.*



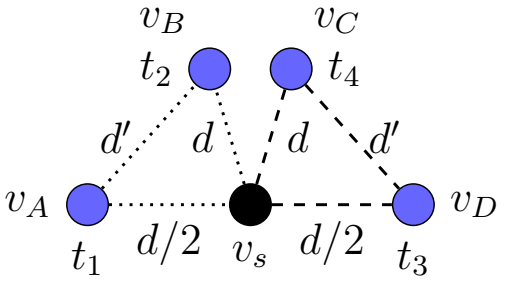
(a) Task locations



(b) Task allocation in Phase 1



(c) Task allocation in Phase 2



(d) Task reallocation during Phase 3

Fig. 3.3. Task allocation in Example 3.3.7

Thus, the allocation after Phase 2 is as shown in Figure 3.3(c). The dotted line shows the tour for agent A_1 and the dashed line shows the tour for agent A_2. The min-max cost for this set of tours is $2d + \epsilon$. In Phase 3, tasks $\{t_1, t_2, t_3, t_4\}$ are deallocated to a set T_1', and a Christofides' tour H' (starting and ending at v_s) is formed on this set. After running TaskSplitter on H', tasks $\{t_1, t_2\}$ are allocated to agent A_1 and tasks $\{t_3, t_4\}$ are allocated to agent A_2. The resulting tour is shown in Figure 3.3(d). The min-max cost for these set of tours is $d + \frac{d}{2} + d' \approx d + \frac{d}{2} + \frac{\sqrt{5}}{2}d \approx 2.62d$. Since the new min-max cost is higher, the previous set of tours from Phase 2 (shown in Figure 3.3(c)) is retained in Phase 3 (line 17 of the HeteroCycleSplit *algorithm).*

3.4 Chapter Summary

In this chapter, we considered the Heterogeneous Agent Cycle Problem (HACP) which aims to allocate tasks to heterogeneous agents subject to agent-task compatibility while minimizing the maximum tour cost. We provided two approximation algorithms to solve HACP. We first proposed a 2β-approximation algorithm CycleSplit, where β is the approximation factor of the algorithm used to split tours. We then use the CycleSplit algorithm to develop a three phase $(5 - \frac{2}{k})$-approximation algorithm (where k is the number of agents available) called HeteroMinMaxCycleSplit that allocates tasks to agents in a "balanced" way.

4. MIN-MAX PATHS FOR TASK ALLOCATION TO HETEROGENEOUS AGENTS

In Chapter 3, we focused on formulating and solving HACP [40]; in this chapter, we consider the Heterogeneous Agent Path Problem (HAPP), which generalizes HACP by relaxing the constraints on the start and end location of tours. The objective of HACP is to find tours for each agent that start and end at a start node (e.g., base), such that the task-agent compatibility constraints are met, and (approximately) minimize the maximum cost incurred by any agent to complete its tours. In contrast, in HAPP, we seek to find a set of *paths* through the tasks (where each path starts at the current location of an agent), where the maximum length of any path is (approximately) minimized. We impose no restrictions on the start/end positions of agents, i.e., agents may start at different locations and are not required to rendezvous back to their start location. In particular, our HAPP formulation enables adaptive and resilient task allocation in real time, where agents are reassigned tasks due to arrival of new tasks or due to failure of certain agents. We use two approaches to solve HAPP, providing a 15-approximation algorithm and a 16-approximation algorithm respectively. The results presented in this chapter were published in [41].

4.1 Problem Statement

Consider a set of tasks T that are required to be completed by a set of k heterogeneous agents $A = \{A_1, A_2, \ldots, A_k\}$. Let v_j denote the start node of agent A_j, $j \in [k]$. Let the set of all start nodes be denoted by $D = \{v_1, v_2, \ldots, v_k\}$. Note that all start locations are not required to be distinct (i.e., start locations are also allowed to be identical), to capture situations where some of the agents start at the same point. We redefine the graph in the problem formulation of HACP in the previous chapter as

follows. Let $G = (V, E)$ be the complete graph with vertex set $V = T \cup D$ and edge set $E = \{(u, v) : u, v \in V,\ u \neq v\}$. Edge weights and travel costs are defined in the same way as in HACP, and distances are assumed to satisfy the triangle inequality. A path is an alternating sequence of vertices and edges which begins and ends with vertices, such that all edges and vertices are distinct. The cost of a path is defined as the sum of the weights of the edges in the path.

Let S_j be the set of tasks allocated to agent A_j. Let $P_j^*(S_j)$ denote the cost of the optimal path on the set $S_j \cup \{v_j\}$ starting from the node v_j. We do not impose any restrictions on the end point of the path. The objective of the Heterogeneous Agent Path Problem (HAPP) is to partition the set of tasks T among agents subject to the task-agent compatibility constraints, such that the maximum path cost among all agents to visit all their allocated tasks from their respective start locations is minimized. The problem is framed as follows.

Heterogeneous Agent Path Problem (HAPP):

$$\begin{aligned} &\min_{S_1, S_2, \ldots, S_k \subseteq T} \ \max_{j \in [k]} \ P_j^*(S_j) \\ &\text{subject to} \quad \cup_{j=1}^{k} S_j = T,\ S_j \cap S_i = \emptyset,\ \forall j \neq i \\ &\qquad\qquad S_j = V_j \cup R_j,\ \forall j \in [k], \\ &\qquad\qquad V_j \subseteq T_{f(j)},\ R_j \subseteq T_0. \end{aligned}$$

The constraints imply that each task must be executed by exactly one agent and that the task set allocated to each agent A_j is a union of type-specific tasks V_j (which is a subset of $T_{f(j)}$, where $f(j)$ is the type of agent A_j) and generic tasks R_j (which is a subset of T_0).

Any instance of the Traveling Salesperson Path Problem (TSPP) [43] can be converted into an instance of HAPP by setting the number of agents as 1, the start location of the salesperson as the start location of the agent, cities as task locations, and all tasks as generic. Thus, HAPP is NP-hard.

We consider two approaches to formulate an approximation algorithm for this problem. In the first approach, we use a solution to a problem known as the Min-

Max Path Cover Problem (MMPCP) [13,44,45],[1] where given a set of k homogeneous agents (with potentially different start locations) and a set of nodes (tasks) to be covered by the agents, paths are found for each agent such that they cover all the nodes while minimizing the maximum cost incurred by any agent. In this approach, we extend the solution to MMPCP to a scenario with heterogeneous agents. In the second approach, we first solve a problem called the Heterogeneous Agent Tree Problem (HATP) and use that solution to solve HAPP. The idea behind this approach is to use trees to find paths.

4.2 Approach 1: Extending MMPCP to HAPP

In this section we develop an algorithm that gives a solution to HAPP by utilizing solutions to the min-max path cover problem (MMPCP). Given a weighted complete graph G, a set of depots D, and a positive integer m, let *MinMaxPathSplit*(G,D,m) be any algorithm that returns a set of m paths, each starting at a distinct node in D, such that the maximum cost of the paths is less than some factor γ of the maximum cost in the optimal set of paths (e.g., [45] provides such an algorithm). We now propose the following algorithm to solve HAPP. As we did with HACP, the general approach will be to first split the type-specific tasks into paths (Phase 1) and then split the generic tasks into paths (Phase 2). The two types of paths are then stitched together for each agent.

Theorem 4.2.1 *The algorithm* HeteroMinMaxPathSplit *is a* 3γ*-approximation algorithm for HAPP, where* γ *is the approximation factor of the MinMaxPathSplit algorithm used in lines 4 and 8 of* HeteroMinMaxPathSplit.

Proof Let V_j and R_j be the set of tasks allocated to agent A_j by *MinMaxPathSplit* in lines 4 and 8 respectively. Let $P_j(V_j)$ be the cost of the path $\mathcal{P}_j^1$ through the nodes in the set V_j starting at node v_j, and let $P_j(R_j)$ be the cost of the path $\mathcal{P}_j^2$ through the

[1] We use the Capacitated Rooted MMPCP considered in [44], where all paths start from a distinct node from the set of start nodes.

Algorithm 5 Min-Max Path by HeteroMinMaxPathSplit Algorithm

procedure HeteroMinMaxPathSplit(A, T, G)

▷ **Phase 1:** Path through type-specific tasks

  for each agent type $i \in [m]$ **do**

    Define $D_i = \{v_j | j \in [k], f(j) = i\}$. Let G_i be the weighted subgraph of G induced by $D_i \cup T_i$.

    Run *MinMaxPathSplit(G_i, D_i, m_i)* to get m_i paths, each rooted at a unique node in D_i.

    For each agent A_j of type i, assign the path starting at node v_j (its start location). Let $\mathcal{P}_j^1$ be the path through type-specific tasks assigned to agent A_j.

  end for

▷ **Phase 2:** Path through generic tasks

  Define $D = \{v_j | j \in [k]\}$. Let G_0 be the weighted subgraph of G induced by $D \cup T_0$.

  Run *MinMaxPathSplit(G_0, D, k)* to get k paths, each rooted at a unique node in D.

  for each agent A_j, $j \in [k]$ **do**

    Assign the path starting at node v_j to agent A_j, $j \in [k]$. Let $\mathcal{P}_j^2$ be the path through generic tasks assigned to agent A_j.

    Join paths $\mathcal{P}_j^1$ and $\mathcal{P}_j^2$ (by first traversing the shorter path among the two and then proceeding to the first node after v_j in the other path) to get the path $\mathcal{P}_j$ for agent A_j.

  end for

  Return the path for each agent.

end procedure

nodes in the set R_j starting at node v_j. In line 11 of HeteroMinMaxPathSplit, paths $\mathcal{P}_j^1$ and $\mathcal{P}_j^2$ are joined (by first traversing the shorter of the two paths and then proceeding to the first node after the start node in the other path) to get the path $\mathcal{P}_j$ for agent A_j. Note that the cost to travel from the end of the shorter path to the start location (i.e., first node of the longer path) cannot exceed the cost of the shorter path, i.e., $\min\{P_j(V_j), P_j(R_j)\}$. Thus, the cost of path $\mathcal{P}_j$ is upper bounded by $P_j(V_j) + P_j(R_j) + \min\{P_j(V_j), P_j(R_j)\}$.

Let $P^*(S_j^*)$ be the cost of the optimal path for agent A_j under the optimal allocation, where S_j^* is the tasks allocated to agent A_j under an optimal allocation. Note that $S_j^* = V_j^* \cup R_j^*$, where V_j^* is the set of type-specific tasks in S_j^* and R_j^* is the set of generic tasks in S_j^*.

The approximation factor $\tilde{R}$ for HeteroMinMaxPathSplit is given by

$$\begin{aligned}
\tilde{R} &\leq \frac{\max\limits_{j\in[k]} \left\{P_j(V_j) + P_j(R_j) + \min\{P_j(V_j), P_j(R_j)\}\right\}}{\max\limits_{j\in[k]} P_j^*(S_j^*)} \\
&\leq \frac{\max\limits_{j\in[k]} P_j(V_j)}{\max\limits_{j\in[k]} P_j^*(S_j^*)} + \frac{\max\limits_{j\in[k]} P_j(R_j)}{\max\limits_{j\in[k]} P_j^*(S_j^*)} + \min\left\{\frac{\max\limits_{j\in[k]} P_j(V_j)}{\max\limits_{j\in[k]} P_j^*(S_j^*)}, \frac{\max\limits_{j\in[k]} P_j(R_j)}{\max\limits_{j\in[k]} P_j^*(S_j^*)}\right\} \qquad (4.1) \\
&\leq \frac{\max\limits_{j\in[k]} P_j(V_j)}{\max\limits_{j\in[k]} P_j^*(V_j^*)} + \frac{\max\limits_{j\in[k]} P_j(R_j)}{\max\limits_{j\in[k]} P_j^*(R_j^*)} + \min\left\{\frac{\max\limits_{j\in[k]} P_j(V_j)}{\max\limits_{j\in[k]} P_j^*(V_j^*)}, \frac{\max\limits_{j\in[k]} P_j(R_j)}{\max\limits_{j\in[k]} P_j^*(R_j^*)}\right\}, \qquad (4.2)
\end{aligned}$$

where inequality (4.1) follows from the max-min inequality $\max_x \min_y f(x,y) \leq \min_y \max_x f(x,y)$, for any function $f(x,y)$ [46]. Inequality (4.2) follows from the fact that V_j^* and R_j^* are subsets of S_j^* and the triangle inequality holds.

Consider an instance of the problem with only the type-specific tasks. Let V_j' be the set of tasks allocated to agent A_j by an optimal algorithm and let $P_j^*(V_j')$ be the cost of the optimal path for agent A_j through the nodes in V_j' starting from node v_j. So,

$$\max_{j\in[k]} P_j^*(V_j') \leq \max_{j\in[k]} P_j^*(V_j^*). \qquad (4.3)$$

Similarly, by considering the instance with only generic tasks, we get

$$\max_{1\leq j\leq k} P_j^*(R_j') \leq \max_{1\leq j\leq k} P_j^*(R_j^*), \qquad (4.4)$$

where $P_j^*(R'_j)$ is the cost of the optimal path for agent A_j from node v_j through the optimal set of tasks R'_j allocated to it in the instance with only generic tasks.

MinMaxPathSplit is a γ-approximation algorithm for the homogeneous agent min-max path problem. Combining this with equations (4.3) and (4.4), we get

$$\max_{j\in[k]} P_j(V_j) \leq \gamma \max_{j\in[k]} P_j^*(V'_j) \leq \gamma \max_{j\in[k]} P_j^*(V_j^*), \tag{4.5}$$

$$\max_{j\in[k]} P_j(R_j) \leq \gamma \max_{j\in[k]} P_j^*(R'_j) \leq \gamma \max_{j\in[k]} P_j^*(R_j^*). \tag{4.6}$$

Substituting equations (4.5) and (4.6) into equation (4.2), we have $\tilde{R} \leq \gamma + \gamma + \min\{\gamma, \gamma\} = 3\gamma$. ■

Corollary 4.2.2 *Using the 5-approximation algorithm for the min-max path cover problem proposed in [45],* HeteroMinMaxPathSplit *is a 15-approximation algorithm for HAPP.*

Proof The proof follows by substituting $\gamma = 5$ into Theorem 4.2.1. ■

The time complexity of Phase 1 of Algorithm 5 is $\mathcal{O}\left(\sum_{i=1}^{m} \hat{\rho}(G_i, m_i)\right)$, where $\mathcal{O}(\hat{\rho}(G_i, m_i))$ is the complexity of *MinMaxPathSplit* that returns m_i rooted paths in G_i. Line 8 of Phase 2 has complexity $\mathcal{O}(\hat{\rho}(G_0, k))$, and lines 9-12 have complexity $\mathcal{O}(k)$. Thus, the overall complexity is $\mathcal{O}\left(\sum_{i=1}^{m} \hat{\rho}(G_i, m_i) + \hat{\rho}(G_0, k) + k\right)$. Using the algorithm proposed in [45] as *MinMaxPathSplit* results in $\hat{\rho}(G, k) = n(|E| + |V| \log |V|) + (k^3 + n^2) \log n$, where n is the number of task nodes in G, k is the number of agents (or start nodes) in G, $|V| = n + k$ is the number of nodes in G and $|E|$ is the number of edges in G. Thus, using this algorithm as *MinMaxPathSplit*, substituting the values of $\hat{\rho}$ (and doing some algebra) gives an overall complexity of $\mathcal{O}(mn(n + k)^2 + mk^3 \log n)$.

4.3 Approach 2: Solution to HAPP Based on Trees

We now provide an alternate method to solve HAPP, where we first find min-max trees for agents and use them as the basis to find min-max paths. A tree is an

acyclic connected graph. The cost of the tree is defined as the sum of the weights of the edges in the tree. Let $F_j^*(S_j)$ denote the cost of the optimal (minimum cost) tree on the set S_j rooted at node v_j, where v_j is the start node of agent A_j. We now formulate the Heterogeneous Agent Tree Problem (HATP) as follows, where the goal is to find an optimal allocation of tasks to agents satisfying the task-agent compatibility constraints, such that the maximum cost among all agents to construct a tree on the tasks allocated to them is minimized.

Heterogeneous Agent Tree Problem (HATP):

$$\begin{aligned} \min_{S_1,S_2,\ldots,S_k \subseteq T} \quad & \max_{j\in[k]} \; F_j^*(S_j) \\ \text{subject to} \quad & \cup_{j=1}^{k} S_j = T, \; S_j \cap S_i = \emptyset, \; \forall j \neq i \\ & S_j = V_j \cup R_j, \; \forall j \in [k], \\ & V_j \subseteq T_{f(j)}, \; R_j \subseteq T_0. \end{aligned}$$

The existing literature on homogeneous min-max tree cover problems looks at generating a tree cover such that the maximum cost among the trees in the tree cover is minimized. Given a graph $G = (V, E)$ and a set of roots $R \subset V$, the min-max R-rooted tree cover problem aims to find a tree cover of G, such that each tree in the tree cover has a distinct root in R. The min-max R-rooted tree cover problem is NP-hard [32]. Let $V \backslash R$ denote the set of nodes in the set V but not in the set R. Any instance of the above problem can be converted into an instance of HATP by assigning all agent types as 1, assigning all nodes in $V \backslash R$ as generic tasks and assigning the set R as the set of depots (or start nodes) D. Thus, HATP is also NP-hard.

A 4-approximation algorithm to the homogeneous min-max R-rooted tree cover problem, called *Rooted-Tree-Cover*, is provided by [32]. We make use of this algorithm with some modifications to solve HATP. In order to explain the modification made to *Rooted-Tree-Cover* and the rationale behind the modification, we provide a short description of *Rooted-Tree-Cover*. The algorithm selects a bin size B (the optimal value of B is found by a binary search) and runs on the inputs R, G, k and B, where

$G = (V, E)$ is an undirected weighted graph, $R \subset V$ denotes the set of roots, and k denotes the number of trees desired in the tree cover. All edges of weight more than B are first removed from G. Next, all the root nodes are contracted to a single new node r'; for each non-root node v, the set of edges from v to root nodes is replaced by a single edge from v to r', with weight equal to the smallest weight from v to any root node. A minimum spanning tree (MST) [9,10] is constructed on the resulting graph. A forest (i.e., a set of disjoint trees) $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_{|R|}\}$ is then obtained from the MST by un-contracting the root nodes (i.e., by splitting r' back into the individual root nodes, and replacing each edge (r', v) in the MST by an edge from v to its closest root node). At this point, we have a set of trees rooted at R; however, there is no guarantee on the size of each tree (and indeed, some trees might be much larger than others). Thus, the next step is to split the trees and rebalance them across the roots.

To do this, [32] provides a procedure termed "edge-decomposition" which takes each tree $\mathcal{J}_i \in \mathcal{J}$ (with root r_i) and a number B, and returns a set of subtrees of $\mathcal{J}_i$, denoted $S^{(i)} = \{S_i^j\}$, each with weight in the interval $[B, 2B)$, along with a leftover tree L_i rooted at r_i, with weight no more than B. These subtrees and leftover tree may have nodes in common, but are edge disjoint.

Edge decomposition and splitting away of subtrees proceeds as follows. For a node $v \in V(\mathcal{J}_i)$, let T_v denote the rooted subtrees hanging from v. Define subtree T_e comprising of T_v, the parent u of the node v and the edge $e(u, v)$. If $w(T_e) < B$, then it is light; if $w(T_e) \in [B, 2B)$, then it is medium, and if $w(T_e) \geq 2B$, then it is heavy. A subtree T_e is split away from the original tree by designating it as a new part, removing edges in T_e from the original tree and keeping only those nodes and edges in the original tree that are connected to its root node. Medium subtrees are split away first. If every subtree is either light or heavy, let v be a heavy node such that all its children are light. Group edges $\{e_1, e_2, \ldots, e_p\}$ from v to its children until the cumulative edge weight of the trees hanging from these edges exceeds B. Then, split away the subtree $\cup_{i=1}^{p} T_{e_i}$. The decomposition ends when the weight of the remaining

tree (called the leftover tree) is less than B. The subtrees may or may not contain the root node, but the leftover tree includes the root node.

A bipartite graph is then constructed with one side as the vertex set R and the other side representing trees in set $\cup_{i=1}^{|R|} S^{(i)}$. An edge is present between nodes representing root r and subtree S_j^i in the bipartite graph if the distance between them is at most B. A maximum matching is performed on the bipartite graph. If all the subtrees are matched, they are connected to the root node according to the matching to obtain the required tree cover, else "failure" is returned for the selected value of B, and the algorithm tries with a higher value of B. The algorithm returns a rooted tree cover (if it is feasible) such that the cost of any tree in the tree cover does not exceed $4B$.[2] A more detailed explanation of the *Rooted-Tree-Cover* algorithm can be found in [32].

4.3.1 Constant Factor Algorithm for HATP

We extend now the work on homogeneous min-max tree cover problems to the case with heterogeneous agents. As we have done with the previous algorithms, we process type-specific tasks and generic tasks separately and then join the resulting subtrees. Let $D = \{v_1, \ldots, v_k\}$ denote the set of start nodes of all agents and $D_i = \{v_j | j \in [k], f(j) = i\}$ denote the set of start nodes of agents of type $i \in [m]$. Recall that the number of agents of type i is m_i, $i \in [m]$, where m is the number of types of agents such that $\sum_{i=1}^{m} m_i = k$, and $f(j)$ is a function that takes an agent number and returns the agent type. Let v_j be the start location of agent A_j, $j \in [k]$. Let $G = (V, E)$ be the complete graph with vertex set $V = T \cup D$ and edge set $E = \{(u, v) : u, v \in V,\ u \neq v\}$. Edge weights are defined in the same way as before. Let w_{max} be the edge of maximum weight in G. Let $w(\cdot)$ denote the cost of a given tree.

[2]Note that the trees returned by *Rooted-Tree-Cover* in [32] may not be node-disjoint; however, since we are assuming that the edge weights satisfy the triangle inequality, one can simply prune the duplicated nodes from the returned trees without increasing the cost of any tree.

Consider a set of tasks T, and a set of start nodes D. Given the graph $G = (V, E)$ with $V = T \cup D$, the set D, and a positive number k, let *MinMaxTreeCover(G, D, k)* be any algorithm that returns k subtrees such that the maximum cost of the subtrees is within a factor δ of the maximum cost of the optimal (min-max cost) set of subtrees; for example, *Rooted-Tree-Cover* from [32] is such an algorithm. Here, we present Algorithm 6 which builds on *Rooted-Tree-Cover* to cater to the problem of meeting the type-compatibility constraints. In particular, we will first allocate type specific tasks to the agents, and then assign generic tasks *while accounting for the existing allocations of type-specific tasks.*

Within Phase 2 of Algorithm 6, lines 8-11 are the same as the steps in *Rooted-Tree-Cover* (as discussed earlier in this chapter; additional details can be found in [32]). The modification incorporated to better cater to a heterogeneous min-max framework is captured by steps 12-14 (and the algorithm AllocateSubtreeToRoot). The *Rooted-Tree-Cover* algorithm finds a maximum matching on the bipartite graph. In contrast, we assign costs to nodes in the bipartite graph, and find two allocations. One is the maximum matching and the second is an allocation of subtrees (in a greedy manner) to root nodes (or agents) while taking into account the cost incurred by the node (agent) if the subtree is allocated to it. We then find which of the two allocations have lower min-max cost in Algorithm 7. If the former has lower weight, and all subtrees are matched to root nodes, we return the maximum matching (as in *Rooted-Tree-Cover*). However, if there is an allocation of subtrees to nodes (by the greedy method) of lower min-max cost than the maximum matching, with cost less than B, then we return that matching. This additional processing improves the performance in certain scenarios, as we will illustrate shortly. In line 15 of Algorithm 6, we check if a valid allocation is returned for the selected value of B. In the last part of the algorithm, we combine the allocation of type-specific tasks (from Phase 1) and generic tasks (line 16). To do this, for each agent, we form an MST on all the tasks allocated to that agent (line 17). Note that the MST is chosen as it has the minimum cost among all trees than span all the tasks allocated to that agent. Algorithm 8

Algorithm 6 Min-Max Tree by HeteroTreeSplit Algorithm

1: **procedure** HeteroTreeSplit(A, T, G, B)

▷ **Phase 1:** Type-specific task allocation

2: **for** each agent type $i \in [m]$ **do**

3: Let $D_i = \{v_i | f(j) = i,\ j \in [k]\}$. Let G_i be the weighted subgraph of G induced by $D_i \cup T_i$.

4: Run *MinMaxTreeCover(G_i, D_i, m_i)* to get m_i subtrees rooted on D_i. If any node is present in multiple subtrees, prune that node from all trees except the tree with the minimum cost.

5: For each agent A_j of type i, assign subtree with start node v_j to that agent.

6: **end for**

▷ **Phase 2:** Generic task allocation

7: Remove nodes corresponding to type-specific tasks from G and remove the edges incident on them. Let the resulting graph be $G' = (V', E')$.

8: Remove all edges with weight more than B.

9: Contract nodes in D to a single node r' such that for all $v_i \in D$ and $v \in V' \backslash D$, an edge (v, v_i) induces an edge (v, r'), where $w((v, r')) = \min_{v_i \in D} w((v, v_i))$. Construct an MST rooted at r' on the resulting graph.

10: Obtain forest $\{\mathcal{J}_i, i \in [k]\}$ by un-contracting root nodes in the MST (by replacing each edge (v, r') with an edge (v, v_i), where $w((v, r')) = w((v, v_i))$ and $v_i \in D$).

11: For each tree $\mathcal{J}_i$ in the forest, edge-decompose it into a set of subtrees $S^{(i)} = \{S^i_j\}$ and a leftover tree L_i using the procedure in [32].

12: Define the set X as nodes representing root nodes in D, and the set Y as nodes representing subtrees in the set $\cup_{i=1}^{k} S^{(i)}$. Let the cost associated with each root node $v_j \in X$ be $w(j)$, which is the sum of the weight of the subtree (consisting of type-specific tasks) allocated to it in line 5 and the leftover tree associated with it in line 11. Let $w(S^i_j)$ be the cost of a subtree $S^i_j \in Y$.

13: Form a weighted bipartite graph $H[X, Y]$ with X and Y as the two vertex sets, where an edge exists between nodes $v \in X$ and $z \in Y$ if and only if the graph contains an edge of weight at most B from the root node v to some node in the subtree represented by z.

14: Run AllocateSubtreeToRoot$(H[X, Y], B)$.

15: If no valid allocation is found, return failure.

16: Else, for each agent A_j, allocate the tree consisting of the node $v_j \in D$, subtrees matched to the node v_j and leftover tree (if any) containing the root v_j.

17: For each agent, find the MST on all the tasks allocated to that agent. If any node is allocated to multiple agents, prune that node from all trees except that of the agent with the minimum tree cost.

18: **end procedure**

Algorithm 7 AllocateSubtreeToRoot$(H[X, Y], B)$ Algorithm

1: **procedure** AllocateSubtreeToRoot$(H[X, Y], B)$

2: Let $c_X(x)$ be the cost of a node $x \in X$, $c_Y(y)$ be the cost of a node $y \in Y$, and $c_e(x, y)$ be the cost of an edge between them (if it exists).

3: Find a **maximum matching** on the bipartite graph. Let this matching be M_1. Let $v_x \in Y$ be the vertex matched to $x \in X$. Define $cost(M_1) = \max_{x \in X} c_X(x) + c_Y(v_x) + c_e(x, v_x)$. If no matching is found, set $cost(M_1) = \infty$.

4: Find a **greedy min-max allocation** as follows. Set all nodes in Y as unallocated.

5: Select an unallocated node $y \in Y$ with maximum cost $c_Y(y)$. Find node $x \in X$ such that $c_X(x) + c_Y(y) + c_e(x, y)$ is minimum. Allocate y to x, and set it as allocated. Update $c_X(x) \leftarrow c_X(x) + c_Y(y) + c_e(x, y)$.

6: Repeat step 5 until all nodes in Y are allocated. Let this allocation be M_2.

7: Define $cost(M_2) = \max_{x \in X} c_X(x)$.

8: If $\min\{cost(M_1),\ cost(M_2)\} < B$, return the minimum cost allocation among M_1 and M_2. If not, return failure.

9: **end procedure**

Algorithm 8 Min-Max Tree by HeteroMinMaxTreeSplit Algorithm

1: **procedure** HeteroMinMaxTreeSplit(A, T, G)
2: Do binary search for B in $[0,\ w_{max}|V|]$ (where $w_{max} = \max_{e \in E} w(e)$) to find the smallest value of B for which HeteroTreeSplit(A, T, G, B) returns a valid set of trees.
3: Return the set of trees returned by HeteroTreeSplit(A, T, G, B).
4: **end procedure**

performs a binary search over the range of values for B, and calls Algorithm 6 in each iteration.

Example 4.3.1 *In this example, we illustrate how the modification we introduced in the Rooted-Tree-Cover algorithm while allocating generic tasks improves the performance in certain classes of problems. Consider a scenario with four agents whose positions are represented by the root nodes r_1, r_2, r_3, and r_4 respectively. Suppose that after allocation of type-specific tasks (lines 2-6 of* HeteroTreeSplit*) and splitting away subtrees (lines 7-12), the costs of the trees associated with each root node are 1, 100, 100, 100 respectively. Consider the bipartite graph H generated in step 13 of the algorithm, with nodes $\{r_1, r_2, r_3, r_4\}$ forming the partition X and nodes $\{S_1, S_2, S_3, S_4\}$ forming the partition Y (Figure 4.1). Recall that each S_i represents a subtree consisting of generic tasks (formed in steps 7-12 of the algorithm). Let the cost associated with nodes S_1, S_2, S_3 and S_4 be 10 and let the edge cost be 1 for all edges.*

The maximum matching M_1 matches r_j to S_j, for $j \in [4]$. The cost associated with node r_1 is 12, and with nodes r_2, r_3 and r_4 is 111. Thus, cost(M_1) = $\max\ \{12, 111, 111, 111\} = 111$. *On the other hand, the greedy min-max allocation M_2 allocates all nodes in the second partition to r_1. The cost associated with r_1 is 45, whereas the cost associated with r_2, r_3, r_4 are 100. Thus, cost(M_2) =* $\max\ \{45, 100, 100, 100\} = 100$. *The Rooted-Tree-Cover algorithm returns M_1 whereas the* HeteroMinMaxTreeSplit *algorithm returns the one with the minimum cost, i.e., M_2 when allocating generic tasks.*

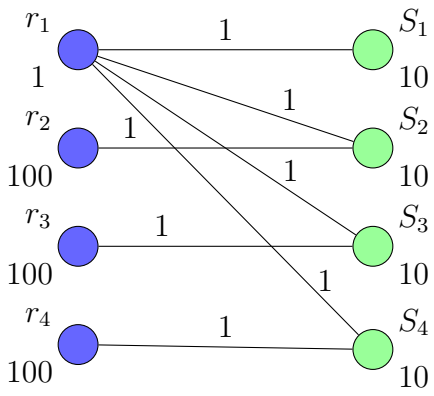


Fig. 4.1. Bipartite graph $H[X, Y]$ with the root nodes in partition X and the subtrees in partition Y

Theorem 4.3.2 HeteroMinMaxTreeSplit *is an 8-approximation algorithm for HATP, when using Rooted-Tree-Cover as MinMaxTreeCover in line 4 of the* HeteroTreeSplit *algorithm.*

Proof Let B be the maximum cost of the trees in the tree cover returned on the set T by HeteroMinMaxTreeSplit. Let B_1 be the maximum cost of the trees in the tree cover on type-specific tasks and let B_2 be the maximum cost of the trees in the tree cover on generic tasks returned by the HeteroMinMaxTreeSplit algorithm. Let B^* be the min-max cost of the optimal tree cover. Also, let B_1^* be the min-max cost of the optimal tree cover on type-specific tasks and let B_2^* be the min-max cost of the optimal tree cover on generic tasks. The approximation factor $\bar{R}$ for HeteroMinMaxTreeSplit is given by

$$\bar{R} = \frac{B}{B^*} \leq \frac{B_1 + B_2}{B^*} = \frac{B_1}{B^*} + \frac{B_2}{B^*} \leq \frac{B_1}{B_1^*} + \frac{B_2}{B_2^*}. \tag{4.7}$$

Since *Rooted-Tree-Cover* is a 4-approximation algorithm, $\frac{B_1}{B_1^*} \leq 4$. Line 14 of HeteroMinMaxTreeSplit (described in the AllocateSubtreeToRoot algorithm) finds the minimum among two allocations. The first method (maximum matching) is the same as in *Rooted-Tree-Cover*, and so the minimum of the two methods is no

worse that the cost returned by *Rooted-Tree-Cover*. Hence, by construction, $\frac{B_2}{B_2^*} \leq 4$. Substituting these values into (4.7), $\bar{R} \leq 8$. ■

Remark 4.3.3 *The greedy allocation in Algorithm 7 is a heuristic that can improve performance in practice, but does not come with a guarantee. One could consider other modifications to the* AllocateSubtreeToRoot *procedure to allocate generic-task subtrees to root nodes (while accounting for the costs of the task-specific subtrees that are already allocated to each root node). Specifically, by viewing each root node as a "parallel machine" and the subtrees as "jobs", the task of allocating subtrees to roots to minimize the maximum cost is an instance of a makespan minimization problem [47]. There are various approximation algorithms and heuristics for such problems, which can be further leveraged in* AllocateSubtreeToRoot *to potentially improve the performance of the algorithm. We leave the investigation of such modifications for future work.*

4.3.2 Using HATP to solve HAPP

In the previous section, we looked at an algorithm that finds min-max trees for heterogeneous agents. In this section, we use it to solve the Heterogeneous Agent Path Problem (HAPP). Let *HeteroMinMaxTreeSplit*(A, T, G) be a ζ-approximation algorithm for HATP. We now present the following algorithm to solve HAPP using a solution of HATP.

Algorithm 9 Min-Max Path by HeteroMinMaxTreeToPath Algorithm

1: **procedure** HeteroMinMaxTreeToPath(A, T, G)
2: Run *HeteroMinMaxTreeSplit*(A, T, G) to get k trees, one for each agent.
3: For each agent A_j, $j \in [k]$, convert its allocated tree to a path by doubling the edges of the tree and shortcutting edges.
4: Return the path for each agent.
5: **end procedure**

Theorem 4.3.4 HeteroMinMaxTreeToPath *is a* 2ζ*-approximation algorithm for HAPP, where* ζ *is the approximation factor of HeteroMinMaxTreeSplit*(A, T, G) *used in line 2 of* HeteroMinMaxTreeToPath.

Proof Let S_j denote the set of tasks allocated to the agent A_j in line 2 of the *HeteroMinMaxTreeSplit*(A, T, G) algorithm. Let S_j^* be the set of tasks allocated to agent A_j under the optimal allocation for HAPP. For agent A_j, $j \in [k]$, let $P_j(\cdot)$ be the cost of the path (starting from the start node v_j of agent A_j) returned by HeteroMinMaxTreeToPath on a given set of task nodes. Let $P_j^*(\cdot)$ be the cost of the optimal path for agent A_j (starting from start node v_j) on a given set of task nodes. Let $F_j^*(\cdot)$ be the cost of the tree (rooted at node v_j) on the set of tasks assigned to agent A_j by *HeteroMinMaxTreeSplit*(A, T, G) in line 2. Without loss of generality, we can assume that the subtrees returned by *HeteroMinMaxTreeSplit*(A, T, G) are Minimum Spanning Trees (MSTs) on the given set of nodes (if not, it is trivial to construct an MST on the set of nodes). Thus, $F_j^*(\cdot)$ is the cost of the MST on the set of task nodes allocated to agent A_j rooted at the start node v_j. The approximation factor $\hat{R}$ for HeteroMinMaxTreeToPath is given by

$$\hat{R} = \frac{\max\limits_{j\in[k]} P_j(S_j)}{\max\limits_{j\in[k]} P_j^*(S_j^*)}. \tag{4.8}$$

In step 3 of Algorithm 9, we double the edges of the MST and shortcut edges to find the path through the set of tasks allocated to each agent. Thus, $P_j(S_j) \leq 2F_j^*(S_j)$. Also, a path is a special instance of a tree and the MST is the tree of minimum cost among all trees, i.e., $F_j^*(S_j^*) \leq P_j^*(S_j^*)$. Substituting these inequalities in (4.8), we get

$$\hat{R} \leq \frac{\max\limits_{j\in[k]} 2F_j^*(S_j)}{\max\limits_{j\in[k]} P_j^*(S_j^*)} \leq 2\frac{\max\limits_{j\in[k]} F_j^*(S_j)}{\max\limits_{j\in[k]} F_j^*(S_j^*)}. \tag{4.9}$$

For any ζ-approximation algorithm that solves HATP,

$$\max_{j\in[k]} F_j^*(S_j) \leq \zeta \max_{j\in[k]} F_j^*(\hat{S}_j) \leq \zeta \max_{j\in[k]} F_j^*(S_j^*), \tag{4.10}$$

where $\hat{S}_j$ is the optimal allocation of tasks to agent A_j for HATP. The second inequality holds as $\hat{S}_j$ is an optimal solution for HATP. From equations (4.9) and (4.10), we get $\hat{R} \leq 2\zeta$. ■

Corollary 4.3.5 HeteroMinMaxTreeToPath *is a 16-approximation algorithm for HAPP if* HeteroMinMaxTreeSplit *is used in line 2.*

Proof The result follows from Theorem 4.3.2 and Theorem 4.3.4. ■

4.4 Comparison of the approaches to solve HAPP

In Sections 4.2 and 4.3, we considered two approaches to solve HAPP. The first approach extends the Min-Max Path Cover Problem (MMPCP), which is analogous to the homogeneous agent version of HAPP. In particular, we showed that using a γ-approximation algorithm to MMPCP, we get a 3γ-approximation algorithm for HAPP. The second approach first finds tree covers for heterogeneous agents and then converts trees to paths. In order to find an appropriate tree cover for the set of heterogeneous agents, we first formulated the Heterogeneous Agent Tree Problem (HATP) and provided a constant factor approximation algorithm to solve it. We then showed that given a ζ-approximation solution for HATP, we can find a 2ζ-approximation algorithm for HAPP. Figure 4.2 shows both the approaches. Which approach is better will depend on the quality of the available approximation algorithms for MMPCP and HATP; in this thesis, the approach via HATP yielded a 16-factor approximation, versus a 15-factor approximation via MMPCP.

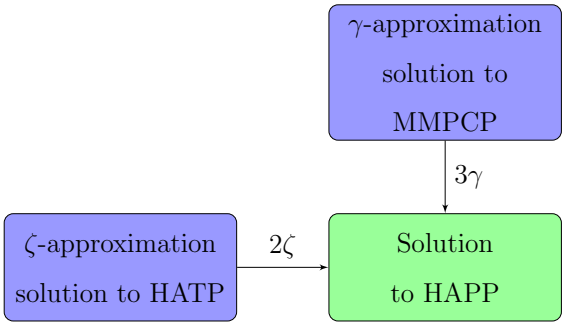


Fig. 4.2. Two solution approaches to HAPP

4.5 Simulations

We provide simulations of the HeteroMinMaxPathSplit algorithm for HAPP. Simulations were carried out on an Intel(R) Core(TM) i7-6700 CPU (3.40 GHz). Task locations were generated uniformly at random within a square defined by $(x, y) \in [-10, 10] \times [-10, 10]$ with the base at the origin. Figure 4.3 shows an example case with 100 tasks (with 60 type-specific tasks and 40 generic tasks) and 6 agents where agent A_1 is of type 1, agents A_2, A_3 are type 2, and the remaining three agents are type 3. The plot shows the paths generated by the HeteroMinMaxPathSplit algorithm.

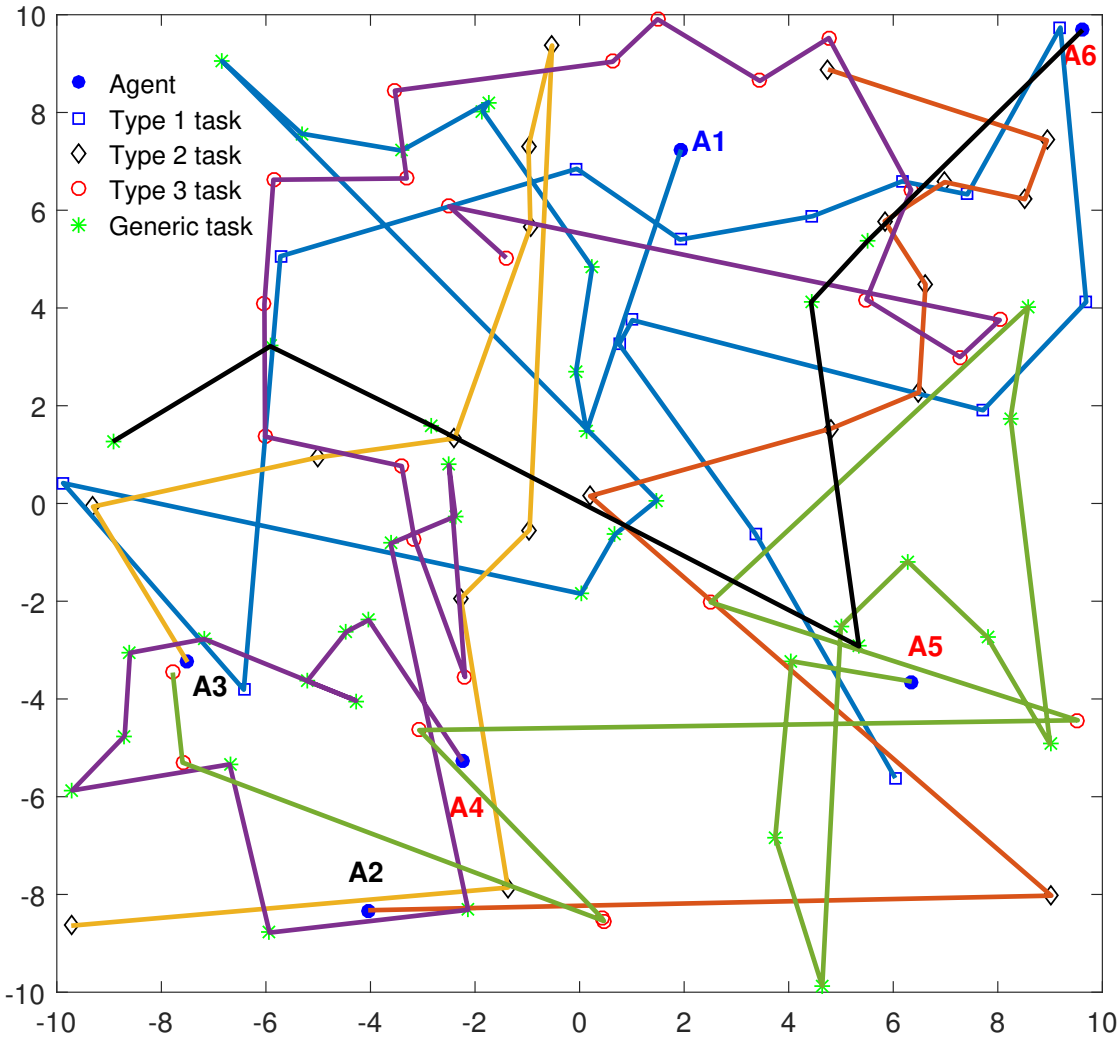


Fig. 4.3. Output of Algorithm 5 on an instance of HAPP with 6 agents and 100 tasks.

In Figure 4.4, we plot the run time versus the size of the instance. The run times are averaged over 100 instances; the number of tasks are varied, while keeping the number of agents and types fixed. As predicted by our complexity analysis, the runtime grows approximately cubicly with the number of tasks.

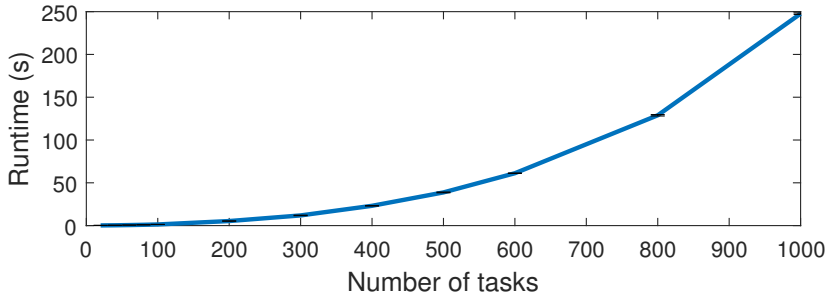


Fig. 4.4. Run time of Algorithm 5 for various sizes of problem instances.

4.6 Chapter Summary

In this chapter, we presented a class of task allocation problem where agents can be heterogeneous in their functionality and the objective is to minimize the maximum travel cost among all agents. We extended the Heterogeneous Agent Cycle Problem (HACP) to the Heterogeneous Agent Path Problem (HAPP), where the maximum cost of paths (instead of tours) for the set of heterogeneous agents is minimized. This framework generalizes HACP and enables adaptive and resilient task allocation in real-time.

We considered two approaches to solve HAPP. The first approach extended the Min-Max Path Cover Problem (MMPCP) to HAPP by splitting HAPP into two instances of MMPCP. We showed that this method yields a 3γ approximation solution to HAPP, where γ is the approximation factor of the algorithm used to solve MMPCP. This approach provides a 15-approximation algorithm for HAPP. In the second approach to solve HAPP, we first formulated the Heterogeneous Agent Tree Problem

(HATP) and proposed a 2ζ-approximation algorithm for it, where ζ is the approximation factor of the algorithm used to find a min-max tree cover (without heterogeneity). This gives an 8-approximation algorithm for HATP. We then converted trees to paths to get a 16-approximation algorithm for HAPP.

5. OPTIMAL POLICIES FOR RISK-AWARE SENSOR DATA COLLECTION BY A MOBILE AGENT

In Chapters 3 and 4, we looked at approaches to route agents in risk-free environments. In this chapter, we look at a scenario where the agent faces a risk of destruction when it travels. Vast improvements in technology over the last few decades have made the use of unmanned systems more accessible and affordable. Such systems are now increasingly used for various application, particularly in hazardous (or hostile) environments. In such applications, it is important to take into consideration the *risk* an agent faces while operating in that environment. Even in benign (or relatively safe) environments, there is some form of risk present, for instance, failures internal to the agent (agent software or hardware malfunctions), environmental risks (such as weather), etc. Thus, in this chapter, we study routing under risk. Examples of such scenarios include reconnaissance missions in hostile territories where an agent is required to collect tactical information, post-disaster search-and-rescue missions where an agent is required to collect information on survivors, etc.

There is a large literature that considers path planning problems without risk to agents [13, 19, 21, 30, 32]. There are relatively much fewer works that consider risk. Risk-based path planning problems in the literature consider problem definitions that consider constraint-based and stochastic risk models. *Constraint-based* approaches [18, 48] model risk as cost on edges. Thus, the total risk of a path would be the sum of the risks of the edges in that path. This approach then tries to find paths that have a risk within a given threshold value. This formulation makes it challenging to consider the trade-offs between task value and risk. In contrast, *stochastic* risk models [49, 50] represent risk as a probabilistic event that affects the outcome and tries to maximize the expected value (or utility) of the solution. The stochastic events in these models affect the cost of the solution but does not consider the possibility of loss of agent. In

this chapter, we consider the problem formulation introduced in [51, 52] which looks at *attrition-based risk*. Unlike the previous risk models, the attrition-based risk model considers the potential loss of agents due to stochastic events. In particular, we focus on the case where a single agent is required to collect sensor measurements and return them to a base. We characterize several key properties of optimal walks for a mobile agent under this scenario. We show that the optimal walk must consist of a set of disjoint cycles. We then define a quantity which we call the *reward-to-risk* ratio. We then characterize the properties and ordering of the cycles in an optimal walk based on their reward-to-risk ratios. We also provide bounds on the length of the cycles in an optimal walk. For the special case when the risk is sufficiently high, we provide the optimal policy. We also provide three heuristics for the general case and compare their performance through simulations. This chapter focuses on risk-aware path planning for a single agent and the results presented in this chapter were published in [53]. We use the insights and approaches developed from this chapter to tackle the multi-agent risk-aware formulation which has significant additional complexities (discussed in Chapter 6).

5.1 Problem Formulation and Notation

We adopt the following formulation from [51, 52]. Consider an agent and a set of n tasks $T = \{t_1, t_2, \cdots, t_n\}$, where each task represents a sensor reading from the task location. Let the agent be located at a base (or start location) denoted by v_s. The tasks in T are dispersed over a geographic area, and need to be collected and returned to the base v_s. To capture this, consider a graph $G = (V, E)$ with vertex set $V = T \cup \{v_s\}$ and edge set $E = \{(u, v) : u, v \in V,\ u \neq v\}$. Let each edge $e = (u, v) \in E$ have a weight $w(e)$ denoting the Euclidean distance between the nodes u and v. Note that the distances satisfy the triangle inequality. We also assume that all distances are non-negative integers (note that rational distances can be converted to integer distances by simply scaling all distances appropriately). The

cost of executing a task (i.e., collecting the sensor measurement) is assumed to be very small compared to travel costs and is hence neglected. Let each task $t_i \in T$ have a positive reward r_i.

Recall that a walk is an alternating sequence of vertices and edges $\{v_0, e_1, v_1, e_2, \ldots, e_k, v_k\}$ that begins and ends with a vertex, and each edge e_i satisfies $e_i = (v_{i-1}, v_i)$. A closed walk is a walk with the same vertex as its starting and ending vertex. Note that nodes can be visited multiple times in a walk, unlike a cycle where each node is visited only once (except for the start node). The length of a walk is the sum of the weights of all the edges in that walk. Given a closed walk W that starts and ends at v_s, we can break it into an ordered set of closed walks $\mathcal{C}(W)$ each starting and ending at node v_s. This can be done by splitting the walk W each time we encounter the node v_s in the walk. In the special case that W visits v_s only at the start and end, the set of walks $\mathcal{C}(W)$ will be a singleton set. Given an ordered set of closed walks $\mathcal{C}(W) = \{W_1, W_2, \ldots, W_k\}$, the agent picks the first walk W_1, starts from the base v_s, visits each of the nodes in the walk, and then returns to v_s. If the set $\mathcal{C}(W)$ has multiple elements, the agent proceeds to do the same for the next walk W_2 and so forth, until it exhausts all the walks in $\mathcal{C}(W)$.

Let the probability of survival for an agent per unit distance be defined as ψ, $0 \leq \psi \leq 1$. Then, the probability that an agent successfully traverses a path of length d is given by ψ^d. Now, consider a closed walk W and its associated set $\mathcal{C}(W)$. Let d_i be the length of the closed walk $W_i \in \mathcal{C}(W)$. For the walk W_1, the probability that the agent successfully completes the walk is given by ψ^{d_1}. If the agent successfully completes a walk and returns to the base, it receives the reward for all tasks it has completed in the walk, otherwise it gets a zero reward. This captures the sensor measurement collection scenario, where the measurements are received (or recorded) only when the agent returns to the base. So, for the walk W_1, the agent gets a reward $R_1 = \sum_{t_j \in W_1} r_j$ with a probability $p_s(W_1) = \psi^{d_1}$, and zero with probability $1 - p_s(W_1)$. Note that the agent cannot get a reward for a task that has already been completed. So, if walk W_2 is successfully completed after W_1 (i.e., the agent returns to v_s after

visiting all of the vertices in W_2), it gets the reward for tasks in W_2 that were not present in W_1. We denote this set of tasks by $W_2 \backslash W_1$. Thus, for the walk W_2, the agent gets a reward $R_2 = \sum_{t_j \in W_2 \backslash W_1} r_j$ with a probability $p_s(W_2) = \psi^{d_1} \times \psi^{d_2} = \psi^{d_1+d_2}$, and a reward of zero with probability $1 - p_s(W_2)$. This follows from the fact that the agent must first successfully complete W_1 before starting W_2, and then must successfully complete W_2. Generalizing, for walk $W_i \in \mathcal{C}(W)$, the agent gets a reward

$$R_i = \begin{cases} \sum_{t_j \in W_i \backslash \cup_{p=1}^{i-1} W_p} r_j, \text{ for } i > 1 \\ \sum_{t_j \in W_1} r_j, \text{ for } i = 1, \end{cases}$$

with a probability $p_s(W_i) = \psi^{d_1+d_2+\ldots+d_i}$, and a reward zero with probability $1 - p_s(W_i)$. Thus, the expected reward of a walk W_i is given by

$$E_0(W_i) = R_i \psi^{\sum_{j=1}^{i} d_j},$$

and the expected reward of an ordered set of closed walks $\mathcal{C}(W)$ is given by

$$E_0(\mathcal{C}(W)) = \sum_{i=1}^{k} R_i \psi^{\sum_{j=1}^{i} d_j}, \tag{5.1}$$

where k is the number of walks in $\mathcal{C}(W)$.

5.1.1 Cost of Loss of Agent

In our discussion so far, we have considered the probability of agent loss, but have not given an associated cost yet. Let the cost of the agent be θ, i.e., if the agent does not survive, then a loss of θ is incurred. The probability that the agent does not survive a set of walks $\{W_1, W_2, \ldots, W_k\}$ is given by $1 - \psi^{\sum_{j=1}^{k} d_j}$. Thus, the expected utility (with agent cost) for a walk W can be written as

$$E(\mathcal{C}(W)) = \sum_{i=1}^{k} R_i \psi^{\sum_{j=1}^{i} d_j} - \theta \left(1 - \psi^{\sum_{j=1}^{k} d_j} \right). \tag{5.2}$$

Note that in the special case where $\theta = 0$ i.e., the agent is expendable, this reduces to (5.1).

Given the start location v_s, a set of tasks T to be completed, the survival probability per unit distance ψ, and a weighted graph $G = (V, E)$ with $V = T \cup \{v_s\}$, we aim to find a walk W^* (and the corresponding set of ordered closed walks $\mathcal{C}(W^*)$) that maximizes the expected utility given by (5.2). Thus, our objective can be written as follows.

Single Agent Risk Aware Task Execution (SARATE):

$$\text{Find} \quad W^* \in \arg\max_{W \in \mathcal{W}} E(\mathcal{C}(W)),$$

where $\mathcal{W}$ is the set of all walks that begin and end at the base (or start location) v_s.

5.2 Characteristics of the Optimal Walks

We now provide certain properties of an optimal walk W^*.

Proposition 5.2.1 *An optimal walk for the SARATE problem consists of a set of disjoint cycles (except at the base v_s, which is common to all cycles).*

Proof To prove this, it suffices to show that no task appears more than once in the optimal walk W^*; this would then imply that the set $\mathcal{C}(W^*)$ will consist of a set of disjoint cycles, with the exception of the base v_s which appears in all cycles.

We show this in two steps. First, we prove that the optimal walk consists of cycles. We then show that these cycles must be disjoint. Let the optimal walk be $\mathcal{C}(W^*) = \{W_1^*, \ldots, W_k^*\}$. We prove both steps by contradiction.

Suppose there exists at least one walk $W_i^* \in \mathcal{C}(W^*)$ that is not a cycle (i.e., it has at least one task node t that is visited more than once). Construct a new walk W_i' from W_i^* by omitting all visits to node t except the first one. We can see that W_i' has the same reward as W_i^*, but has a shorter walk length than W_i^* (due to the triangle inequality). Hence, the expected reward of the walk W_i' is higher than that

of the walk W_i^*. By replacing W_i^* with W_i' in $\mathcal{C}(W^*)$, we obtain a new walk W' that has higher utility than W^*, which contradicts the optimality of $\mathcal{C}(W^*)$. Thus, the optimal walk consists of a set of cycles.

The second part can be proved by a similar argument by showing that if the optimal cycles are not disjoint, then a set of cycles with higher expected utility can be obtained by removing the tasks that are repeated (from all cycles except where the task first occurs).

We now show that these cycles must be disjoint. Suppose the walks in $\mathcal{C}(W^*) = \{W_1^*, \ldots, W_k^*\}$ are not disjoint at a node other than v_s. Then there exist a pair of walks W_i^* and W_j^* that have a common task node t, and $i \neq j$. Without loss of generality, let $i < j$ (meaning the walk W_i^* is ordered before W_j^*). Now, if we remove the task node from the walk W_j^*, we get a new walk W_j' with the same reward as W_j^*, but with shorter walk length. Thus, the expected reward of the walk W_j' is higher than that of the walk W_j^*. Thus, we can find a set of walks $\mathcal{C}(W') = \{W_1^*, \ldots, W_{j-1}^*, W_j', W_{j+1}^*, \ldots, W_k^*\}$ with a higher expected utility than $\mathcal{C}(W^*)$, which violates the assumption of optimality of $\mathcal{C}(W^*)$.

■

Based on the above proposition, in order to solve the SARATE Problem, we need to find the optimal way to split the set of tasks into cycles and find the order in which these cycles are traversed, based on the survival probability ψ and the locations of the tasks.

5.2.1 Ordering of the Cycles

In the previous section, we showed that the optimal policy consists of a set of disjoint cycles. Even if we have a set of disjoint (except at v_s) cycles, the expected reward depends on the order in which we select the cycles, since the probability of earning the reward from a given cycle depends on surviving all the other cycles traversed before it. We now characterize this order for the optimal walk.

Definition 5.2.1 *Consider an ordered set of cycles* $\mathcal{C}(W) = \{W_1, W_2, \ldots, W_k\}$*, such that each cycle starts and ends at node* v_s*. Let the reward and length of each cycle* $W_i \in \mathcal{C}(W)$ *be* R_i *and* d_i *respectively. For each cycle* $W_i \in \mathcal{C}(W)$*, define its* ***reward-to-risk ratio*** γ_i *as*

$$\gamma_i = \frac{R_i \psi^{d_i}}{1 - \psi^{d_i}}. \tag{5.3}$$

Proposition 5.2.2 *Let* W^* *be an optimal walk for the SARATE problem, with* $\mathcal{C}(W^*) = \{W_1^*, W_2^*, \ldots, W_k^*\}$*, such that each cycle starts and ends at node* v_s*, and the cycles are disjoint (except at* v_s*). Let* R_i *and* d_i *be the total reward and length, respectively, of the cycle* W_i^**. Let the survival probability be* $0 < \psi < 1$*. Then, the walks in the ordered set* $\mathcal{C}(W^*)$ *will be in non-increasing order of their reward-to-risk ratio, i.e., if* $i < j$*, then* $\gamma_i \geq \gamma_j,\ \forall i, j \in \{1, \ldots, k\}$*.*

Proof We prove the result by contradiction. Suppose the optimal walk W^* is such that the cycles in $\mathcal{C}(W^*) = \{W_1^*, W_2^*, \ldots, W_k^*\}$ do not follow a non-increasing ordering of their reward-to-risk ratio. Then, there exists a pair of consecutive walks W_i^* and W_{i+1}^* such that $\gamma_i < \gamma_{i+1}$. Using the definition of γ_i in (5.3), we have

$$\begin{aligned} & R_i \psi^{d_i} - R_i \psi^{d_i + d_{i+1}} < R_{i+1} \psi^{d_{i+1}} - R_{i+1} \psi^{d_i + d_{i+1}} \\ \implies & R_{i+1} \psi^{d_{i+1}} + R_i \psi^{d_i + d_{i+1}} > R_i \psi^{d_i} + R_{i+1} \psi^{d_i + d_{i+1}}. \end{aligned} \tag{5.4}$$

From (5.2), the expected utility of W^* is given by

$$E(\mathcal{C}(W^*)) = \sum_{i=1}^{k} R_i \psi^{\sum_{j=1}^{i} d_j} - \theta \left(1 - \psi^{\sum_{j=1}^{k} d_j} \right). \tag{5.5}$$

Construct a new walk W' by swapping W_i^* and W_{i+1}^* in $\mathcal{C}(W^*)$. Thus, $\mathcal{C}(W') = \{W_1^*, \ldots, W_{i-1}^*, W_{i+1}^*, W_i^*, W_{i+2}^*, \ldots, W_k^*\}$. The expected utility of W' is given by

$$\begin{aligned} E(\mathcal{C}(W')) = R_1 \psi^{d_1} + \ldots + R_{i-1} \psi^{\sum_{j=1}^{i-1} d_j} \\ + R_{i+1} \psi^{\sum_{j=1}^{i-1} d_j + d_{i+1}} + R_i \psi^{\sum_{j=1}^{i+1} d_j} + R_{i+2} \psi^{\sum_{j=1}^{i+2} d_j} \end{aligned}$$

$$+ \ldots + R_k \psi^{\sum_{j=1}^{k} d_j} - \theta\Big(1 - \psi^{\sum_{j=1}^{k} d_j}\Big). \tag{5.6}$$

Subtracting (5.5) from (5.6), we get

$$\begin{aligned} E(\mathcal{C}(W')) - E(\mathcal{C}(W^*)) &= R_{i+1}\psi^{\sum_{j=1}^{i-1} d_j + d_{i+1}} + R_i \psi^{\sum_{j=1}^{i+1} d_j} - \Big(R_i \psi^{\sum_{j=1}^{i} d_j} + R_{i+1}\psi^{\sum_{j=1}^{i+1} d_j}\Big) \\ &= \psi^{\sum_{j=1}^{i-1} d_j}\Big(R_{i+1}\psi^{d_{i+1}} + R_i \psi^{d_i + d_{i+1}} \\ &\quad - \big(R_i \psi^{d_i} + R_{i+1}\psi^{d_i + d_{i+1}}\big)\Big). \end{aligned}$$

Now, applying (5.4), we obtain $E(\mathcal{C}(W')) - E(\mathcal{C}(W^*)) > 0$, which contradicts the optimality of W^*. ■

5.2.2 Skipping Tasks

Recall that the agent has a value θ that is incurred as a cost if the agent does not survive. Thus, there may be cases where it is better to forgo certain tasks rather than risk losing the agent to visit those tasks. We now formalize the condition under which a subset of cycles in an ordered set of cycles will be included or omitted by the agent.

Proposition 5.2.3 *Every cycle in the optimal walk will have a reward-to-risk ratio greater than or equal to θ.*

Proof We prove by contradiction. Suppose the optimal walk $\mathcal{C}(W^*) = \{W_1^*, W_2^*, \ldots, W_k^*\}$ contains a cycle with reward-to-risk ratio less than θ. Without loss of generality, let this cycle be W_k^* since the cycles are ordered such that the reward-to-risk ratio $\gamma_i \geq \gamma_j$ if $i < j$; recall from Proposition 5.2.2 that the optimal walk satisfies this ordering. Let R_i be the reward that can be obtained from cycle W_i^*, and d_i be its length. The expected utility for traversing $\mathcal{C}(W^*)$ is given by (5.5).

Consider a set of cycles $\mathcal{C}(W')$ obtained by removing W_k^* from $\mathcal{C}(W^*)$. The expected utility for $\mathcal{C}(W')$ is given by

$$E(\mathcal{C}(W')) = \sum_{i=1}^{k-1} R_i \psi^{\sum_{j=1}^{i} d_j} - \theta\left(1 - \psi^{\sum_{j=1}^{k-1} d_j}\right). \tag{5.7}$$

Subtracting (5.5) from (5.7), we get

$$\begin{aligned} E(\mathcal{C}(W')) - E(\mathcal{C}(W^*)) &= \theta\Big(\psi^{\sum_{j=1}^{k-1} d_j} - \psi^{\sum_{j=1}^{k} d_j}\Big) - R_k\psi^{\sum_{j=1}^{k} d_j} \\ &= \psi^{\sum_{j=1}^{k-1} d_j}\big(1-\psi^{d_k}\big)\Big(\theta - \frac{R_k\psi^{d_k}}{1-\psi^{d_k}}\Big) \\ &> 0, \end{aligned}$$

where the last inequality follows since $\frac{R_k\psi^{d_k}}{1-\psi^{d_k}} = \gamma_k < \theta$.

Thus, $E(\mathcal{C}(W')) > E(\mathcal{C}(W^*))$, which contradicts the optimality of $\mathcal{C}(W^*)$. Thus, every cycle in the optimal walk will have a reward-to-risk ratio greater than or equal to θ. ■

In Proposition 5.2.3, we showed that all cycles in the optimal walk will have sufficiently high reward-to-risk ratio. We now focus on tasks rather than cycles. For each task, we define its reward-to-risk ratio as the reward-to-risk ratio of the cycle containing just that task. In other words, for a task t, construct a cycle $W_t = \{v_s, e(v_s,t), t, e(t,v_s), v_s\}$ and compute its reward-to-risk ratio over this cycle. If the length of W_t is d_t and reward of task t is r_t, then the reward-to-risk ratio of task t is given by $\gamma_t = \frac{r_t\psi^{d_t}}{1-\psi^{d_t}}$.

Proposition 5.2.4 *Every task with reward-to-risk ratio greater than θ will be included in the optimal walk.*

Proof We prove by contradiction. Suppose the optimal walk $\mathcal{C}(W^*) = \{W_1^*, \ldots, W_k^*\}$ does not include a task t with reward-to-risk ratio higher than θ. Let the length of cycle W_i^* be d_i and let the reward of cycle W_i^* be R_i, for $i \in \{1, \ldots, k\}$. The expected utility of W^* is given by (5.5).

Let the reward of task t be r_t. Construct a cycle $W_t = \{v_s, e(v_s,t), t, e(t,v_s), v_s\}$, and let its length be d_t. Consider the walk $\mathcal{C}(W') = \{W_1^*, \ldots, W_k^*, W_t\}$ obtained by appending the cycle W_t to the optimal walk. The expected utility of this walk is given by

$$E(\mathcal{C}(W')) = \sum_{i=1}^{k} R_i\psi^{\sum_{j=1}^{i} d_j} + r_t\psi^{\sum_{j=1}^{k} d_j + d_t} - \theta\left(1 - \psi^{\sum_{j=1}^{k} d_j + d_t}\right). \tag{5.8}$$

Subtracting (5.5) from (5.8), we obtain (after some algebra)

$$E(\mathcal{C}(W')) - E(\mathcal{C}(W^*)) = \psi^{\sum_{j=1}^{k} d_j}(1-\psi^{d_t})\Big(\frac{r_t\psi^{d_t}}{1-\psi^{d_t}} - \theta\Big),$$

which is positive since $\gamma_t = \frac{r_t\psi^{d_t}}{1-\psi^{d_t}} > \theta$. This contradicts the optimality of $\mathcal{C}(W^*)$, thus proving the result. ■

We have shown that any task with sufficiently high reward-to-risk ratio will be included in the optimal walk. It is worth noting that the converse does not hold (a task with low reward-to-risk ratio could get included as part of a cycle that has a high reward-to-risk ratio).

We show this through the following example.

Example 5.2.1 *Consider a scenario where an agent located at base v_s has three tasks $\{t_1, t_2, t_3\}$ to complete. Let the value θ of the agent be 10. Each task is located at a distance of 100 units from the base, and the distance between tasks t_2 and t_3 is 1 unit as shown in Figure 5.1. Let the probability of survival per unit distance be 0.99. The reward of task t_1 is 40 and the reward of tasks t_2 and t_3 are both 35. For this set of values, $\gamma_{t_1} = 6.1883$ and $\gamma_{t_2} = \gamma_{t_3} = 5.4148$, i.e., the reward-to-risk ratio of each of the tasks is less than the value of the agent. Thus, by Proposition 5.2.3, none of the tasks are individually worth completing (the expected utility will be negative).*

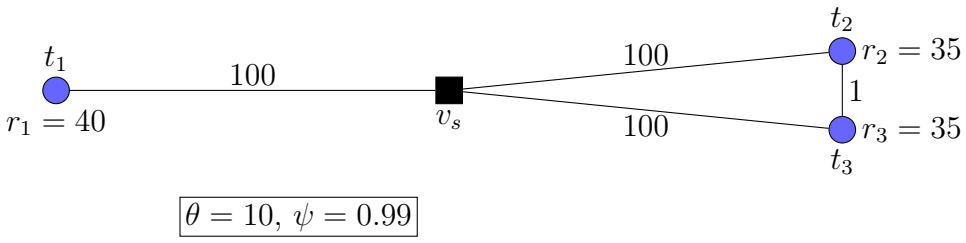


Fig. 5.1. Task locations in Example 5.2.1

However, if we were to consider a cycle containing tasks $\{t_2, t_3\}$, then the reward-to-risk ratio of this cycle is 10.7047, which is greater than the value of the agent. Note

that in this example, the optimal walk is in fact the walk from v_s through tasks t_2 and t_3 back to the base v_s (t_1 does not get executed in the optimal walk despite having the highest reward-to-risk ratio). Thus, tasks t_2 and t_3 will get executed in the optimal walk as part of the cycle containing $\{t_2, t_3\}$ even though their individual reward-to-risk ratios are less than θ.

5.2.3 Bound on Length of Cycles

We now provide a result showing that the cycles in the optimal walk cannot be too much longer than the distance from the base to the closest task in that walk and back. In particular, we will provide an upper bound on how much longer the cycle could be, as a function of the survival probability ψ, the rewards, and the agent cost θ.

Definition 5.2.2 *Consider an instance of the SARATE Problem, where tasks $T = \{t_1, \ldots, t_n\}$ are to be carried out by an agent (of value θ) located at base v_s. Let the reward of task $t_i \in T$ be r_i. Define $\bar{d}$ as*

$$\bar{d} = \log_\psi \frac{\min\limits_{i \in \{1,\ldots,n\}} r_i}{\sum\limits_{i=1}^{n} r_i + \theta}, \tag{5.9}$$

where ψ is the probability of survival of the agent per unit distance.

Theorem 5.2.2 *Consider an agent of value θ located at base v_s. Let W^* be the optimal walk on tasks $T = \{t_1, \ldots, t_n\}$ for this agent, with $\mathcal{C}(W^*) = \{W_1^*, \ldots, W_k^*\}$. Denote the length of walk $W_i^* \in \mathcal{C}(W^*)$ by d_i. Then, $\forall i \in \{1, 2, \ldots, k\}$,*

$$d_i \leq 2d_{i,min} + \bar{d}, \tag{5.10}$$

where $\bar{d}$ is defined in (5.9) and $d_{i,min} = \min\limits_{t_j \in W_i^*} d(v_s, t_j)$.

Proof We prove by contradiction. Suppose that there exists at least one walk $W_i^* \in \mathcal{C}(W^*)$ such that (5.10) does not hold. Consider the first such walk. Without loss of generality, let this walk be W_1^*. Then,

$$d_1 > 2d_{1,min} + \bar{d}. \tag{5.11}$$

Let the reward of walk W_i^* be R_i. The expected utility of $\mathcal{C}(W^*)$ is given by (5.5). Let t_p be the task closest to v_s on W_1^*, and let its reward be r_p. Split the cycle W_1^* into two cycles - one cycle W_1' with only task node t_p, and the other cycle W_2' with all the other nodes in W_1^* (except the node t_p that is closest to v_s). Note that the length of the walk W_1' is $2d_{1,min}$ by construction. Denote the length of W_2' by d_1' (where $d_1' \leq d_1$). Construct a new walk $\mathcal{C}(W') = \{W_1', W_2', W_2^*, \ldots, W_k^*\}$, by replacing W_1^* with W_1' and W_2'. The expected utility of this walk is given by

$$\begin{aligned} E(\mathcal{C}(W')) = r_p\psi^{2d_{1,min}} &+ (R_1 - r_p)\psi^{2d_{1,min}+d_1'} \\ &+ R_2\psi^{2d_{1,min}+d_1'+d_2} + \ldots + R_k\psi^{2d_{1,min}+d_1'+\sum_{i=2}^{k} d_i} \\ &- \theta(1 - \psi^{2d_{1,min}+d_1'+\sum_{i=2}^{k} d_i}). \end{aligned}$$

Thus,

$$\begin{aligned} E(\mathcal{C}(W')) - E(\mathcal{C}(W^*)) &= r_p\psi^{2d_{1,min}} \\ &\quad + (R_1 - r_p)\psi^{2d_{1,min}+d_1'} + R_2\psi^{2d_{1,min}+d_1'+d_2} \\ &\quad + \ldots + R_k\psi^{2d_{1,min}+d_1'+\sum_{i=2}^{k} d_i} + \theta\psi^{2d_{1,min}+d_1'+\sum_{i=2}^{k} d_i} \\ &\quad - R_1\psi^{d_1} - \ldots - R_k\psi^{\sum_{i=1}^{k} d_i} - \theta\psi^{\sum_{i=1}^{k} d_i} \\ &> r_p\psi^{2d_{1,min}} - R_1\psi^{d_1} - \ldots - R_k\psi^{\sum_{i=1}^{k} d_i} - \theta\psi^{\sum_{i=1}^{k} d_i} \\ &> r_p\psi^{2d_{1,min}} - R_1\psi^{d_1} - \ldots - R_k\psi^{d_1} - \theta\psi^{d_1} \\ &\geq r_p\psi^{2d_{1,min}} - \Big(\sum_{i=1}^{n} r_i + \theta\Big)\psi^{d_1} \\ &= \psi^{2d_{1,min}}\Big(r_p - \Big(\sum_{i=1}^{n} r_i + \theta\Big)\psi^{d_1-2d_{1,min}}\Big). \end{aligned} \tag{5.12}$$

From (5.11), since $d_1 - 2d_{1,min} > \bar{d}$, we have

$$\psi^{d_1-2d_{1,min}} < \psi^{\bar{d}} = \frac{\min\limits_{i\in\{1,\ldots,n\}} r_i}{\sum\limits_{i=1}^{n} r_i + \theta} \quad \text{(from (5.9))}$$

$$\implies r_p \geq \min_{i\in\{1,\ldots,n\}} r_i > \Big(\sum_{i=1}^{n} r_i + \theta\Big)\psi^{d_1-2d_{1,min}}$$

$$\implies r_p - \Big(\sum_{i=1}^{n} r_i + \theta\Big)\psi^{d_1-2d_{1,min}} > 0.$$

Substituting this into (5.12), we see that $E(\mathcal{C}(W')) - E(\mathcal{C}(W^*)) > 0$. This contradicts the optimality of the walk W^*, thus proving the theorem. ■

5.3 Optimal Walks for Sufficiently Small ψ

Based on our result in Theorem 5.2.2, we now show that when the probability of survival of the agent (per unit distance) is very small, the optimal walk consists of a set of cycles that each contain just a single task. We refer to these as "one-shot" cycles as the agent covers exactly one task in each cycle. In other words, when the agent faces a high risk of being destroyed, it is best for the agent to focus on a single task at a time. To prove this, we will start with the following result.

Corollary 5.3.1 *Consider an agent of value θ located at base v_s, and a set of tasks $T = \{t_1, \ldots, t_n\}$, where the reward of each task t_i is given by r_i. Define the graph $G = (V, E)$ with $V = T \cup \{v_s\}$, and $E = \{(u, v)|u, v \in V, u \neq v\}$, where the edge length between two nodes is given by the distance between them. Define d_{min} as the shortest edge length in G. If the survival probability per unit distance ψ satisfies*

$$\psi^{d_{min}} < \frac{\min\limits_{i\in\{1,\ldots,n\}} r_i}{\sum\limits_{i=1}^{n} r_i + \theta}, \tag{5.13}$$

then the optimal walk consists of a set of one-shot cycles.

Proof Let the optimal walk be given by the cycles $\mathcal{C}(W^*) = \{W_1^*, W_2^*, \ldots, W_k^*\}$. Let the length of the cycle W_i^* be given by d_i. We prove the result by contradiction. Assume that at least one cycle in $\mathcal{C}(W^*)$ is not a one-shot cycle. Without loss of generality, this can be taken as W_1^*. Let t_p be the task closest to the base v_s in W_1^*, and let $d_{1,min}$ be its distance from v_s. The cycle W_1^* is strictly longer than the one-shot cycle containing t_p. In particular, $d_1 - 2d_{1,min} \geq d_{min}$ as d_{min} is the minimum

edge length in the graph. However, from (5.13), $d_{min} > \bar{d}$, where $\bar{d}$ is defined in (5.9). Thus, the cycle W_1^* violates the condition (5.10) given in Theorem 5.2.2, and so cannot be in the optimal walk, providing the desired contradiction. ■

Theorem 5.3.2 *Suppose the probability of survival per unit distance satisfies* (5.13)*. Then, the optimal walk can be found with an algorithm whose runtime increases polynomially with the number of nodes.*

Proof Let $T = \{t_1, \ldots, t_n\}$ be the set of tasks to be covered by the agent located at base v_s. From Corollary 5.3.1, we know that the optimal walk consists of a set of one-shot cycles. Form a set of one-shot cycles $\mathcal{C}(W) = \{W_1, \ldots, W_n\}$ such that $W_i = \{v_s, e(v_s, t_i), t_i, e(t_i, v_s), v_s\}$. Sort the cycles in non-increasing order of their reward-to-risk ratio (defined by (5.3)), and let W' be the output of the sorting algorithm (which is polynomial-time in the number of nodes ([54])). For each cycle in W', if the reward-to-risk ratio is greater than θ (the agent's value), then by Proposition 5.2.4 and Corollary 5.3.1, that cycle will be included in the optimal walk. Else, if the reward-to-risk ratio of a cycle in W' is less than θ, then by Proposition 5.2.3, that cycle will be omitted from the optimal walk. Thus, by removing all one-shot cycles from W' that have reward-to-risk ratio less than θ, we obtain the optimal solution. ■

5.4 Evaluation of Heuristics for SARATE

In the previous sections we characterized certain properties of the optimal solutions for SARATE, and provided the optimal policy for the special case where ψ is sufficiently small. We now investigate three heuristic algorithms for general instances of the problem.

Sequential Greedy (SG) Algorithm: In this algorithm, the tasks are first arranged in decreasing order of their reward-to-risk ratio. We start with no cycles. For each task, find the gain in expected utility if the task were to be inserted into an existing cycle (by evaluating the utility by inserting the task between each pair

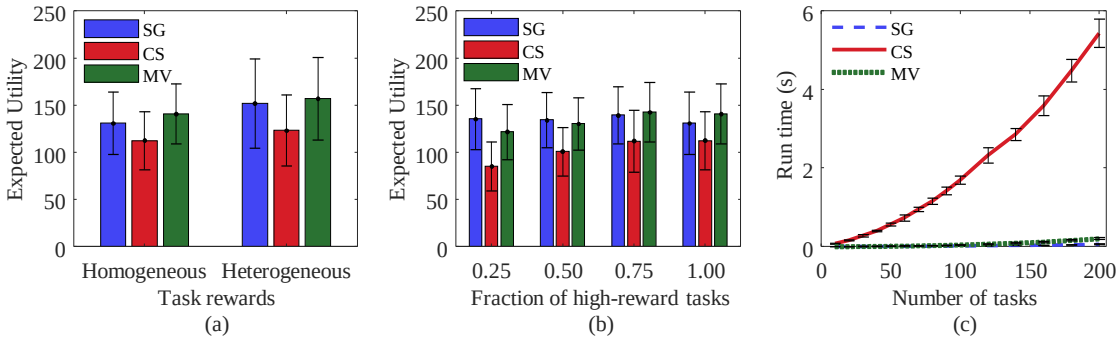


Fig. 5.2. Comparison of performance of different heuristic algorithms: (a) Expected utility when tasks have homogeneous rewards (50) versus heterogeneous rewards (between 1 and 99), (b) Expected utility as the fraction of high reward tasks vary, (c) Run time as the number of tasks vary.

of consecutive nodes in each existing cycle and choosing the best such location) or added as a new cycle (from the base to the task and back). If the maximum gain in expected utility is positive, then add the task to the corresponding cycle (or form a new cycle, if that yields a larger utility). Order the formed cycles based on their reward-to-risk ratio and keep only those cycles with reward-to-risk ratio greater than θ.

Cycle Split (CS) Algorithm: In this algorithm, a tour is formed on the set of all tasks and this tour is split into k cycles (each starting and ending at the base) using the k-*SPLITOUR* algorithm by [30] for $k \in \{1, \ldots, n\}$. The algorithm k-*SPLITOUR* returns a set of cycles for each value of k, such that the maximum cost of any cycle is (approximately) minimized. For each set of cycles, compute the expected utility by ordering the cycles based on their reward-to-risk ratio and considering only those cycles with reward-to-risk ratio greater than θ. The set of subtours with the highest expected utility is selected. To find the initial tour in our simulations, we first found a minimum spanning tree, doubled the edges and shortcutted them.

Markovian (MV) Algorithm: All tasks are marked active and are arranged in non-increasing order of their reward-to-risk ratios. The task t with the highest

reward-to-risk ratio among active tasks is selected. A cycle is formed from base to t and back, and t is marked inactive. For each remaining active task, check if adding that active task to this cycle increases the expected utility by inserting that task node between every two consecutive nodes on the cycle and computing the gain in expected utility; if so add the task t' which maximally increases the expected utility to the cycle (at the appropriate position) and mark it as inactive. Repeat until no more tasks can be added to the cycle, at which point the cycle is completed. Repeat the above steps by choosing the next active task with the highest reward-to-risk ratio, until all tasks are inactive. Order the formed cycles based on their reward-to-risk ratio and consider only those cycles with reward-to-risk ratio greater than θ.

We simulated the above algorithms to study their relative performance. Task locations were generated uniformly at random within a square defined by $(x, y) \in [-100, 100] \times [-100, 100]$ with the base at the origin, and the survival probability per unit distance ψ was set as 0.99. All three algorithms were run on each instance generated. The results shown in Fig. 5.2 are averaged over 100 runs. Define tasks with reward-to-risk ratio higher than θ as high-reward tasks. Fig. 5.2(a) compares the performance of the policies when all 100 tasks have the same reward (set to 50) against the case where different tasks have heterogeneous rewards (generated as integers from a discrete uniform distribution on $[1, 99]$). The agent value θ was selected to be 0.001 less than the lowest reward-to-risk ratio among all tasks so that all tasks are high-reward tasks. Fig. 5.2(b) shows the performance when the fraction of high-reward tasks in a set of 100 tasks is varied (the rewards are generated as integers from a discrete uniform distribution on $[1, 99]$). In order to set the fraction of high-reward tasks, the tasks were arranged in decreasing order of reward-to-risk ratio and θ was set as 0.001 less than the reward-to-risk ratio of the $[nf]^{th}$ task, where n is the number of tasks, f is the desired fraction of high-reward tasks and $[\cdot]$ denotes the nearest integer function. Fig. 5.2(c) shows how the run time scales with the number of tasks while keeping the rewards as 50 for all tasks (θ is set as 0.001 less than the lowest reward-to-risk ratio among all tasks).

Based on our experiments, we observe that the run times for both the Sequential Greedy and the Markovian algorithms are relatively small (even for settings with 200 tasks), while the run time for the CycleSplit algorithm increases more rapidly. Furthermore, the performance of the Sequential Greedy algorithm and the Markovian algorithm are comparable in general. Sequential Greedy performs better when the fraction of high-reward tasks is low, and the Markovian algorithm performs better when the fraction of high-reward tasks is very high. This can be attributed to the fact that when the fraction of high-reward tasks is low, the Markovian algorithm (which assigns the best remaining task to the current tour) myopically assigns each low reward task. On the other hand, when the Sequential Greedy algorithm gets to the low-reward tasks, it evaluates the benefit of adding that task to *each* cycle that has already been formed. This increases the chances that a low-reward task would be added to a cycle that contains other tasks that are located close to that low-reward task, thereby making the relatively low reward of that task worth the risk. As expected, the CS algorithm does not perform as well as the the MV and SG algorithms since CS does not explicitly consider the risk of carrying out each task, and only finds the best way to split a given tour into subtours. Thus, we see that heuristics that incorporate the risk into their decision-making can outperform those that do not.

To understand how well the three algorithms perform against the optimal walk, we implemented a brute force (BF) algorithm to consider all possible sets of disjoint cycles for a given set of tasks. For $n = 7$ tasks (which is the largest number we could evaluate due to the computational complexity of the BF algorithm), we generated 100 instances of high-reward task locations (as described above), and for each instance, compared the performance of the three heuristics and the optimal solution provided by BF. We found that MV performed no worse than 62.0% of the optimal and was within 93.6% of the optimal on average, and SG was no worse than 61.8% and within 96.2% on average. In contrast, CS performed within 60.7% of the optimal on average, but had a worst case performance of 11.8%. While this experiment only considers

instances with a small number of nodes, it indicates that SG and MV are promising algorithms for further investigation.

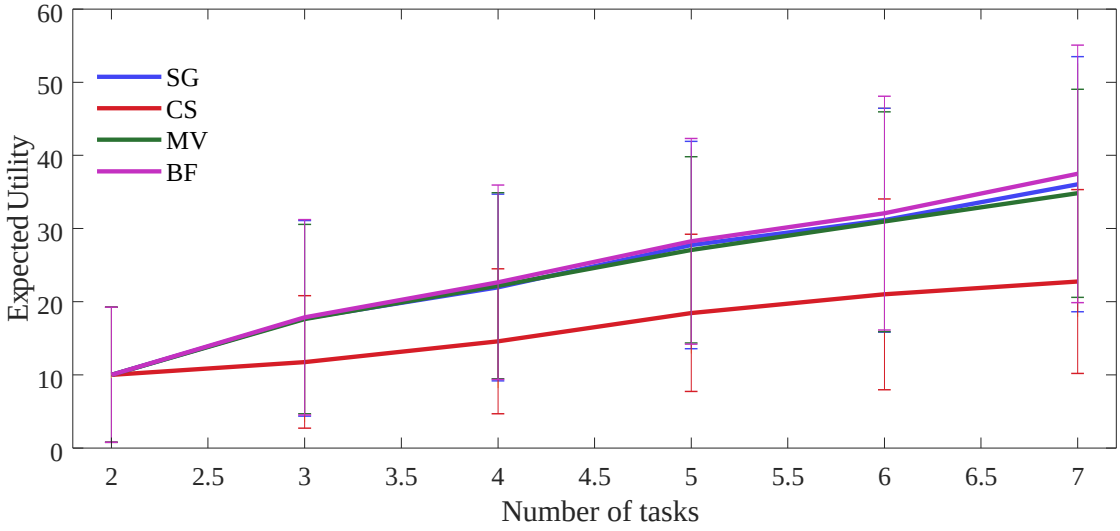


Fig. 5.3. Comparison of performance of different algorithms to the optimal (brute force) solution.

5.5 Chapter Summary

We considered the Single Agent Risk Aware Task Execution (SARATE) problem where an agent is required to collect sensor measurements and return to a base, but faces the risk of destruction as it travels. We showed several key properties of the optimal tours for agents. In particular, we defined a quantity we call the *reward-to-risk ratio* for cycles and tasks. We showed that the ordering of cycles in an optimal walk based on the reward-to-risk ratio. We provided an upper bound on the length of the optimal walk. For the special case where the risk is very high, we explicitly found the optimal policy. For the general case, we provided heuristic policies and evaluated their performance through simulations.

6. POLICIES FOR RISK-AWARE SENSOR DATA COLLECTION BY HETEROGENEOUS MOBILE AGENTS

In this chapter, we extend our analysis from Chapter 5 to multiple agents. Specifically, we look at a heterogeneous agent task allocation problem, where multiple agents are available to collect the sensor measurements and return to the base. The Collection Planning Problem with Attrition Risk (CPPAR) in [51, 52] first looked at this problem, providing a clustering based heuristic called Progressive Risk-aware Clustering (PRC). In this work, we extend the formulation for SARATE problem discussed in Chapter 5 to multiple agents. We show that in high risk scenarios, the scoring scheme is submodular and thus we can provide 0.5-approximation guarantee for a greedy algorithm.

6.1 Problem Formulation and Notation

Consider a set of m agents $A_1, A_2, \ldots, A_m$ located at a base (or start location) v_s. We allow the agents to be heterogeneous (i.e., different agents may have distinct values and survival probabilities per unit distance travelled). Specifically, let the value of agent A_i be denoted by $\theta_i \in \mathbb{R}_{\geq 0}$ and let its survival probability per unit distance be $\psi_i \in (0, 1)$, for $i \in \{1, 2, \ldots, m\}$. As before, let $T = \{t_1, t_2, \ldots, t_n\}$ be the set of tasks to be executed jointly by these m agents, and let the reward obtained for executing task $t_i \in T$ be given by $r_i \in \mathbb{R}_{>0}$. Once again, we will represent the geographic dispersion of the tasks and base via a graph $G = (V, E)$ with $V = T \cup \{v_s\}$, and $E = \{(u, v) | u, v \in V, u \neq v\}$, where the edge length between two nodes is given by the distance between them.

As in the SARATE problem, we wish to allocate the tasks to the agents in such a way as to maximize the reward from the tasks that are successfully completed, while

balancing the risks of agent failures. In particular, we will focus on a particular class of *simultaneous dispatch* policies, where all agents begin executing their specified paths simultaneously, and each task is allocated to at most one agent.[1] Let $\mathcal{T}$ be the set of all such allocations of the tasks in T among the agents that satisfies the partition policy defined above. In particular, each element $\mathcal{A} \in \mathcal{T}$ can be written as $\mathcal{A} = \{T_1, T_2, \ldots, T_m, T_\emptyset\}$, where $T_i,\ i \in \{1, \ldots, m\}$ denotes the set of tasks allocated to agent A_i and $T_\emptyset$ denotes the set of unallocated tasks (i.e., tasks that are omitted by all agents). Then, for each allocation $\mathcal{A} \in \mathcal{T}$, we have $\cup_{i=1}^{m} T_i \cup T_\emptyset = T$, and $T_i \cap T_j = \emptyset$ whenever $i \neq j$.

Let us consider an agent A_i and its walk through the set of tasks T_i allocated to it. Let $\mathcal{W}(T_i)$ denote the set of all closed walks starting and ending at the base v_s that visit all the nodes in T_i exactly once (note that the base v_s may be visited multiple times in each walk), and does not visit nodes in $T_j, \forall\ j \neq i$. For a given walk $W \in \mathcal{W}(T_i)$, let $\mathcal{C}(W) = \{W_1, \ldots, W_p\}$ denote the ordered set of cycles obtained by splitting the closed walk W into cycles each time it visits the node v_s. Let the sum of the rewards of the tasks in the cycle W_i be R_i and let the length of the cycle be d_i. Then, the expected utility of an agent A_i (of value θ_i and probability of survival per unit distance ψ_i) that does the tasks in the set T_i through the set of ordered walks $C(W),\ W \in \mathcal{W}(T_i)$ is given by

$$E_i(\mathcal{C}(W)) = \sum_{j=1}^{p} R_j \psi_i^{\sum_{k=1}^{j} d_k} - \theta_i \left(1 - \psi_i^{\sum_{k=1}^{p} d_k}\right). \tag{6.1}$$

Note that this is the same as the utility (5.2) that we considered in the Chapter 5, rewritten to show the explicit dependence on the index i of the agent under consideration. The score obtained by an agent A_i that is allocated a set of tasks T_i in an optimal walk would then be defined as

$$v_i(T_i) = \max_{W \in \mathcal{W}(T_i)} E_i(\mathcal{C}(W)). \tag{6.2}$$

[1]It is not clear at this point whether this class of policies is, in fact, optimal for the problem we are considering, but we will restrict attention to this class in order to gain insights.

Note, in particular, that for any given allocation T_i of tasks to agent i, the above optimization problem is equivalent to solving the SARATE problem for that agent (considering only the tasks in T_i). The total expected utility of all the agents is given by the sum of the expected utilities of all the agents. Among all the possible allocations $\mathcal{A} \in \mathcal{T}$, we wish to find the allocation $\mathcal{A}^*$ that maximizes the sum of the expected utilities of all the agents. Thus, the problem of interest is as follows.

Multi-Agent Risk-Aware Task Allocation (MARATA):

$$\text{Find} \quad \mathcal{A}^* \in \arg\max_{\mathcal{A}\in\mathcal{T}} \sum_{i=1}^{m} v_i(T_i). \tag{6.3}$$

Remark 6.1.1 *Our formulation of both MARATA and SARATE problems allows us to handle the appearance of new tasks (at a later time). Each time an agent returns to the base, we can find the solution to the current allocation (or execution) problem and then give the agent an updated solution.*

Remark 6.1.2 *Both the SARATE and MARATA problems fall under the ST-SR-TA class of problems based on the taxonomy for multirobot systems given in [7], i.e., each robot (or agent) can execute one task at a time (Single-Task robot), each task can be executed by a single robot (Single-Robot task) and each agent has a Time-extended Assignment of tasks to be executed.*

Given that we can calculate $v_i(T_i)$ for any given set of tasks T_i explicitly and in polynomial time when the walks executed by the agent are guaranteed to be one-shot cycles (by Theorem 5.3.2), we will make the following assumption in the rest of this section.

Assumption 6.1.3 *We will assume that for each agent A_i, $i \in \{1, 2, \ldots, m\}$, its probability of survival per unit distance ψ_i and its value θ_i satisfy the condition* (5.13)*, i.e.,*

$$\psi_i^{d_{min}} < \frac{\min\limits_{j\in\{1,\ldots,n\}} r_j}{\sum\limits_{j=1}^{n} r_j + \theta_i}, \tag{6.4}$$

where d_{min} is the shortest edge length in the graph G.

This assumption (by Corollary 5.3.1) means that the risk faced by each agent is sufficiently high that the optimal walk for each agent will be a set of one-shot cycles through its set of assigned tasks. We will also extend the notion of the reward-to-risk ratio of a task (Def. 5.2.1) to be agent-specific, i.e., the *reward-to-risk ratio of a task t_j with respect to agent A_i* is given by

$$\frac{r_j \psi_i^{d_j}}{1 - \psi_i^{d_j}},$$

where d_j is the round-trip distance from the base v_s to task t_j. Since we know that each agent A_i will execute its assigned tasks as one-shot cycles (under Assumption 6.1.3), and only tasks that have reward-to-risk ratio (with respect to A_i) larger than θ_i will be executed by that agent (by Prop. 5.2.3), we will also make the following assumption in the rest of this section, without loss of generality.

Assumption 6.1.4 *For each task t_j, $j \in \{1, 2, \ldots, n\}$, there is some agent A_i, $i \in \{1, 2, \ldots, m\}$ such that t_j has reward-to-risk ratio (with respect to agent A_i) larger than θ_i.*

We start by showing that the individual agent utility functions are *submodular* in such high-risk scenarios.

6.1.1 Submodularity of the Agent Utilities in High-Risk Scenarios

We start by defining the following notion for each task/agent pair.

Definition 6.1.1 *Consider a task t_j with reward r_j and round-trip distance d_j from the base. Then, the **Maximal Marginal Gain** (MMG) obtained by allocating this task to an agent A_i of value θ_i and probability of survival per unit distance travelled ψ_i is defined as*

$$\mu_{ij} = r_j \psi_i^{d_j} - \theta_i (1 - \psi_i^{d_j}). \tag{6.5}$$

The reason for calling the above quantity the maximum marginal gain will be apparent shortly. Using the MMG for each agent/task pair, we show that the agent utility (6.2) can be written in an alternate form.

Lemma 6.1.5 *Suppose the risk is sufficiently high so that all tasks are executed as one-shot cycles (i.e.,* (6.4) *is satisfied for agent A_i). Then for any set of tasks T_i, and walk $W \in \mathcal{W}(T_i)$, the expected utility* (6.1) *can be written as*

$$E_i(\mathcal{C}(W)) = \sum_{j=1}^{p} \psi_i^{\sum_{k=1}^{j-1} d_k} \mu_{ij} \tag{6.6}$$

where $\mathcal{C}(W) = \{W_1, W_2, \ldots, W_p\}$ is the ordered set of one-shot cycles containing tasks $\{t_1, \ldots, t_p\} \subseteq T_i$ (in that order), μ_{ij} is the MMG of allocating task t_j to agent A_i, and d_k denotes the round-trip distance of task t_k from the base.

Proof Consider the walk $W \in \mathcal{W}(T_i)$, consisting of one-shot cycles through the ordered set of tasks $\{t_1, t_2, \ldots, t_p\}$. Let r_i be the reward and d_i be the round-trip distance from the base for each task t_i. Then, expanding (6.1),

$$\begin{aligned}
E_i(C(W)) &= r_1\psi_i^{d_1} + \ldots + r_p\psi_i^{\sum_{k=1}^{p} d_k} - \theta_i\Big(1 - \psi_i^{\sum_{k=1}^{p} d_k}\Big) \\
&= r_1\psi_i^{d_1} + \ldots + r_p\psi_i^{\sum_{k=1}^{p} d_k} - \theta_i\Big(1 - \psi_i^{d_1} + \psi_i^{d_1} \\
&\quad + \ldots - \psi_i^{\sum_{k=1}^{p-1} d_k} + \psi_i^{\sum_{k=1}^{p-1} d_k} - \psi_i^{\sum_{k=1}^{p} d_k}\Big) \\
&= [r_1\psi_i^{d_1} - \theta_i(1 - \psi_i^{d_1})] + \psi_i^{d_1}[r_2\psi_i^{d_2} - \theta_i(1 - \psi_i^{d_2})] \\
&\quad + \ldots + \psi_i^{\sum_{k=1}^{p-1} d_k}[r_p\psi_i^{d_p} - \theta_i(1 - \psi_i^{d_p})].
\end{aligned}$$

Using the definition of μ_{ij} from (6.5), the above equation yields (6.6). ■

Corollary 6.1.6 *Suppose the risk is sufficiently high so that all tasks are executed as one-shot cycles (i.e.,* (6.4) *is satisfied for agent A_i). Consider any set of tasks $T_i \subseteq T$. Let $\{t_1, t_2, \ldots, t_p\} \subseteq T_i$ be the largest ordered set of tasks satisfying*

$$\frac{r_1\psi_i^{d_1}}{1 - \psi_i^{d_1}} \geq \frac{r_2\psi_i^{d_2}}{1 - \psi_i^{d_2}} \geq \ldots \geq \frac{r_p\psi_i^{d_p}}{1 - \psi_i^{d_p}} > \theta_i,$$

where d_j is the round-trip distance from the base to task t_j, for $j \in \{1, 2, \ldots, p\}$. Then, the utility function (6.2) *can be written as*

$$v_i(T_i) = \sum_{j=1}^{p} \psi_i^{\sum_{k=1}^{j-1} d_k} \mu_{ij}, \tag{6.7}$$

where μ_{ij} is the MMG of assigning task t_j to agent A_i.

Proof We note from Corollary 5.3.1 and Theorem 5.3.2 that in high-risk scenarios, the optimal walk for agent A_i through a set of tasks T_i consists of a set of one-shot cycles, ordered such that the corresponding tasks are in non-increasing order of their reward-to-risk ratios. Furthermore, all tasks with reward-to-risk ratios larger than the agent cost will be included in the optimal walk. Combining this with Lemma 6.1.5, we obtain that the expression (6.2) is given by (6.7). ■

The above result shows that the utility function $v_i(T_i)$ has the form of a time-discounted scoring scheme (described in [20]), where $\psi_i < 1$ is the discounting factor, and $\sum_{k=1}^{j-1} d_k$ is the time taken by agent i to reach task t_j in the sequence $t_1, t_2, \ldots, t_p$. It was argued in [20] that such functions satisfy a property termed *diminishing marginal gain* (similar to the notion of submodularity described below, but defined for ordered sets). Here, we will show formally that the utility function is submodular in high-risk scenarios. We start by recalling the definition of submodularity [55, 56].

Definition 6.1.2 *Consider a scoring scheme $v(\cdot)$ that takes as input a set of tasks and outputs a positive real number (the score). This scoring scheme is* ***submodular*** *if given any set of tasks T, and any tasks $t, \bar{t} \notin T$,*

$$v(T \cup \{t\}) - v(T) \geq v(T \cup \{\bar{t}, t\}) - v(T \cup \{\bar{t}\}). \tag{6.8}$$

Proposition 6.1.1 *Suppose the risk is sufficiently high so that all tasks are executed as one-shot cycles, i.e.,* (6.4) *holds. Then the scoring scheme provided by* (6.2) *is submodular.*

Proof To show that (6.2) is submodular, we need to show that Definition 6.1.2 holds. In Corollary 6.1.6, we showed that when the risk is sufficiently high so that all tasks are executed as one-shot cycles, the scoring scheme given by (6.2) can be written as (6.7). Thus, it suffices to show that (6.7) is submodular. Let the set of tasks allocated to agent A_i be T_i. Let $\{W_1, \ldots, W_p\}$ denote the ordered set of one-shot cycles that maximizes the objective, where walk W_j is a one-shot cycle of length d_j consisting of task t_j. Then,

$$v_i(T_i) = \mu_{i1} + \psi_i^{d_1}\mu_{i2} + \ldots + \psi_i^{d_1+\ldots+d_{p-1}}\mu_{ip}.$$

Consider adding task $t \notin T_i$ to T_i. We assume that t is a high-reward task, since it will not get executed otherwise, and the desired condition will hold trivially. Since all tasks are executed as one-shot cycles, tasks t will be added as a one-shot cycle. Let t be added as a one-shot cycle between W_{k-1} and W_k (for some $k \in \{1, 2, \ldots, |p|+1\}$) in order to maximize the resulting expected utility. Let d_t be the round-trip distance from the base to task t, and let μ_{it} be its MMG. Then,

$$\begin{aligned} v_i(T_i \cup \{t\}) = \mu_{i1} + \psi_i^{d_1}\mu_{i2} + \ldots + \psi_i^{d_1+\ldots+d_{k-2}}\mu_{i(k-1)} \\ + \psi_i^{d_1+\ldots+d_{k-1}}\mu_{it} + \psi_i^{d_1+\ldots+d_{k-1}+d_t}\mu_{ik} \\ + \ldots + \psi_i^{d_1+\ldots+d_{p-1}+d_t}\mu_{ip}. \end{aligned}$$

Thus, the marginal gain in adding task t to the set T_i, denoted MG_t, is given by

$$\begin{aligned} MG_t &= v_i(T_i \cup \{t\}) - v_i(T_i) \\ &= \psi_i^{d_1+\ldots+d_{k-1}}\mu_{it} + (\psi_i^{d_t} - 1)(\psi_i^{d_1+\ldots+d_{k-1}}\mu_{ik} + \ldots + \psi_i^{d_1+\ldots+d_{p-1}}\mu_{ip}). \end{aligned}$$

Similarly, consider adding a task $\bar{t}$ to T_i. Let $\bar{t}$ be added as a one-shot cycle between W_{l-1} and W_l (for some $l \in \{1, 2, \ldots, |p|+1\}$) in order to maximize the resulting expected utility. Let $d_{\bar{t}}$ be the round-trip distance from the base to task $\bar{t}$, and let $\mu_{i\bar{t}}$ be its MMG. Then,

$$v_i(T_i \cup \{\bar{t}\}) = \mu_{i1} + \psi_i^{d_1}\mu_{i2} + \ldots + \psi_i^{d_1+\ldots+d_{l-2}}\mu_{i(l-1)}$$

$$+ \psi_i^{d_1+\ldots+d_{l-1}}\mu_{i\bar{t}} + \psi_i^{d_1+\ldots+d_{l-1}+d_{\bar{t}}}\mu_{il}$$
$$+ \ldots + \psi_i^{d_1+\ldots+d_{p-1}+d_{\bar{t}}}\mu_{ip}.$$

Let us first consider the case where task t has a lower reward-to-risk ratio than $\bar{t}$, and is therefore inserted after $\bar{t}$ in the ordered list, i.e., $l < k$. Then

$$\begin{aligned} v_i(T_i \cup \{t, \bar{t}\}) = \mu_{i1} + \psi_i^{d_1}\mu_{i2} + \ldots + \psi_i^{d_1+\ldots+d_{l-2}}\mu_{i(l-1)} \\ + \psi_i^{d_1+\ldots+d_{l-1}}\mu_{i\bar{t}} + \psi_i^{d_{\bar{t}}}\Big(\psi_i^{d_1+\ldots+d_{l-1}}\mu_{il} + \ldots \\ + \psi_i^{d_1+\ldots+d_{k-2}}\mu_{i(k-1)}\Big) + \psi_i^{d_{\bar{t}}}\psi_i^{d_1+\ldots+d_{k-1}}\mu_{it} \\ + \psi_i^{d_{\bar{t}}+d_t}\Big(\psi_i^{d_1+\ldots+d_{k-1}}\mu_{ik} + \ldots + \psi_i^{d_1+\ldots+d_{p-1}}\mu_{ip}\Big). \end{aligned}$$

Thus, the marginal gain, denoted $MG_{t,1}$, in adding task t to the set $T_i \cup \{\bar{t}\}$ when t is positioned after $\bar{t}$ is given by

$$\begin{aligned} MG_{t,1} &= v_i(T_i \cup \{t, \bar{t}\}) - v_i(T_i \cup \{\bar{t}\}) \\ &= \psi_i^{d_{\bar{t}}}\psi_i^{d_1+\ldots+d_{k-1}}\mu_{i\bar{t}} + (\psi_i^{d_t} - 1)\psi_i^{d_{\bar{t}}}\big(\psi_i^{d_1+\ldots+d_{k-1}}\mu_{ik} + \ldots + \psi_i^{d_1+\ldots+d_{p-1}}\mu_{ip}\big) \\ &= \psi_i^{d_{\bar{t}}}MG_t < MG_t. \end{aligned}$$

Thus, the scoring function satisfies (6.8) in this case. In order to complete the proof, we now look at the case where the task t is ordered ahead of $\bar{t}$, i.e., $k \leq l$. In this case,

$$\begin{aligned} v_i(T_i \cup \{t, \bar{t}\}) = \mu_{i1} + \psi_i^{d_1}\mu_{i2} + \ldots + \psi_i^{d_1+\ldots+d_{k-2}}\mu_{i(k-1)} \\ + \psi_i^{d_1+\ldots+d_{k-1}}\mu_{it} + \psi_i^{d_t}\Big(\psi_i^{d_1+\ldots+d_{k-1}}\mu_{ik} + \ldots \\ + \psi_i^{d_1+\ldots+d_{l-2}}\mu_{i(l-1)}\Big) + \psi_i^{d_t}\psi_i^{d_1+\ldots+d_{l-1}}\mu_{i\bar{t}} \\ + \psi_i^{d_{\bar{t}}+d_t}\Big(\psi_i^{d_1+\ldots+d_{l-1}}\mu_{il} + \ldots + \psi_i^{d_1+\ldots+d_{p-1}}\mu_{ip}\Big). \end{aligned}$$

The marginal gain, denoted $MG_{t,2}$, in adding task t to the set $T_i \cup \{\bar{t}\}$ in this scenario is given by

$$MG_{t,2} = v_i(T_i \cup \{t, \bar{t}\}) - v_i(T_i \cup \{\bar{t}\})$$

$$= \psi_i^{d_1+\ldots+d_{k-1}}\mu_{it} + \Big(\psi_i^{d_t}-1\Big)\Big(\psi_i^{d_1+\ldots+d_{k-1}}\mu_{ik} + \ldots + \psi_i^{d_1+\ldots+d_{l-2}}\mu_{i(l-1)}\Big) + \Big(\psi_i^{d_t}-1\Big)\Big(\psi_i^{d_1+\ldots+d_{l-1}}\mu_{i\bar{t}}\Big) + \psi_i^{d_{\bar{t}}}Big(\psi_i^{d_t}-1\Big)\Big(\psi_i^{d_1+\ldots+d_{l-1}}\mu_{il} + \ldots + \psi_i^{d_1+\ldots+d_{p-1}}\mu_{ip}\Big).$$

Taking the difference of the two marginal gains, we get

$$MG_{t,2} - MG_t = \left(\psi^{d_t}-1\right)\left(\psi^{d_1+\ldots+d_{l-1}}\mu_{i\bar{t}}\right) + \left(\psi^{d_t}-1\right)\left(\psi^{d_{\bar{t}}}-1\right)\left(\psi^{d_1+\ldots+d_{l-1}}\mu_{il} + \ldots + \psi^{d_1+\ldots+d_{p-1}}\mu_{ip}\right) \tag{6.9}$$

$$= \underbrace{\left(\psi^{d_t}-1\right)}_{<0}\underbrace{\psi^{d_1+d_2+\ldots+d_{l-1}}}_{>0}\Big[\mu_{i\bar{t}} + \psi^{d_{\bar{t}}}\left(\mu_{il} + \ldots + \psi^{d_l+d_{l+1}+\ldots+d_{p-1}}\mu_{ip}\right) - \left(\mu_{il} + \ldots + \psi^{d_l+d_{l+1}+\ldots+d_{p-1}}\mu_{ip}\right)\Big]. \tag{6.10}$$

Note that the quantity in the square brackets is simply the marginal gain of adding task $\bar{t}$ to the set of one-shot cycles $\bar{W} = \{W_l, W_{l+1}, \ldots, W_p\}$; task $\bar{t}$ will be placed as a one-shot cycle prior to W_l (as it was when being added to the full set of tasks) since the reward-to-risk ratio of task $\bar{t}$ is at least as large as that of task t_l. The marginal gain of adding $\bar{t}$ to $\bar{W}$ will be positive, since $\bar{t}$ is a high-reward task, i.e.,

$$\mu_{i\bar{t}} + \psi^{d_{\bar{t}}}\left(\mu_{il} + \psi^{d_l}\mu_{i(l+1)} + \ldots + \psi^{d_l+\ldots+d_{p-1}}\mu_{ip}\right) > \mu_{il} + \psi^{d_l}\mu_{i(l+1)} + \ldots + \psi^{d_l+\ldots+d_{p-1}}\mu_{ip}$$

Substituting this into (6.10), we see that $MG_{t,2} - MG_t < 0$. Thus, the scoring scheme satisfies (6.8) in this scenario as well.

Since T_i, t and $\bar{t}$ were arbitrary, the above analysis shows that the function $v_i(\cdot)$ is submodular in high-risk scenarios. ■

We showed that in the instances of our problem where the risk is sufficiently high (so that all tasks are executed as one-shot cycles), our scoring scheme given in (6.2) can be written as (6.7) and is submodular. It is important to note that without the high risk assumption, tasks are no longer guaranteed to be executed as one-shot

cycles and consequently, submodularity may not hold. We illustrate this through the following example.

Example 6.1.7 *In this example, all tasks are high-reward tasks, however the value of ψ is chosen to be high (implying low risk). Therefore tasks are no longer guaranteed to be executed as one-shot cycles. Consider an agent with $\psi = 0.99$ and $\theta = 40$. Consider tasks $\{t_1, t_2, t_3\}$ with rewards 20, 5 and 5 respectively located at distances $3, 4$ and 5 units from the base v_s as shown in Figure 6.1.*

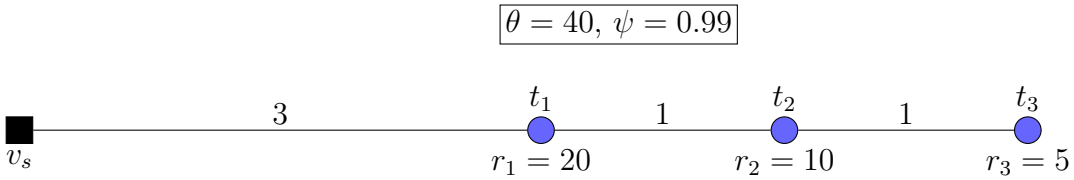


Fig. 6.1. Task locations in Example 6.1.7

Consider the set of tasks $\{t_1, t_2\}$. The walk that executes both the tasks in the same cycle maximizes the expected utility for this set of tasks when using (6.2) *as the scoring scheme. Similarly, for the set of tasks $\{t_1, t_3\}$, the walk that executes both the tasks in the same cycle maximizes the expected utility. The marginal gain in expected utility from adding t_3 to the set $\{t_1\}$ is* 3.7799*. Let us now consider the set of tasks $\{t_1, t_2, t_3\}$. In this example, the walk that executes all three tasks in the same cycle maximizes the expected utility (as opposed to splitting the tasks across multiple cycles). The marginal gain in adding t_3 to the set $\{t_1, t_2\}$ is* 3.9710*, which is higher than the marginal gain of adding t_3 to the set $\{t_1\}$. The key insight here is that the task t_3 is worth more when the agent is already visiting t_2 (since the agent is already close to t_3 at that point), but is worth less if the agent is not already visiting t_2. Thus in this example, submodularity does not hold.*

6.1.2 Approximation Algorithms

Proposition 6.1.1 showed that each agent A_i's function $v_i(\cdot)$ is submodular under high-risk scenarios. Furthermore, it is easy to see that v_i is monotone (i.e., adding new tasks to the argument in v_i will not decrease the score), and that $v_i(\emptyset) = 0$. Thus, the MARATA problem (6.3) becomes an instance of a *submodular welfare* problem [56] (or, more generally, a submodular maximization problem over a *partition matroid* [55]). Such problems have been widely studied in the literature on combinatorial optimization, and there are a variety of algorithms to solve them near-optimally. For example, the techniques in [57–59] provide solutions that are guaranteed to be within a factor $1 - \frac{1}{e}$ of the optimal solution to the problem.

Here, we will discuss a specific greedy algorithm that provides a looser guarantee (namely solutions that are within a factor $\frac{1}{2}$ of optimal), but that is extremely simple to implement. In particular, we will first provide the general algorithm, and then show that this algorithm has an intuitive structure when we consider the MARATA problem with homogeneous agents.

Greedy Algorithm

As argued in [56], the submodular welfare problem admits a simple greedy algorithm that guaranteed to be within a factor $\frac{1}{2}$ of optimal. Since the MARATA problem is an instance of the submodular welfare problem under the high risk scenario (as argued above), we present the greedy algorithm in the context of the MARATA problem as Algorithm 10, with the following result that follows immediately from [56].

Theorem 6.1.8 *Suppose that for each agent $A_i, i \in \{1, 2, \ldots, m\}$, its probability of survival per unit distance ψ_i and its value θ_i satisfy the condition* (6.4)*. Let $\mathcal{A}^* = \{T_1^*, T_2^*, \ldots, T_m^*, T_\emptyset^*\}$ be an optimal allocation to the agents for the MARATA problem*

Algorithm 10 Greedy Algorithm for the MARATA Problem in High-Risk Scenarios

1: **procedure** SGA-M(A, T, G)
2: Initialize $T_1 = T_2 = \cdots = T_m = \emptyset$.
3: Let $t_1, t_2, \ldots, t_n$ be the set of tasks, in any order.
4: **for** j from 1 to n **do**
5: Let $i \in \{1, 2, \ldots, m\}$ be the index of the agent for which $v_i(T_i \cup \{t_j\}) - v_i(T_i)$ is largest (breaking ties arbitrarily).
6: Set $T_i \leftarrow T_i \cup \{t_j\}$.
7: **end for**
8: **end procedure**

(6.3). *Then the greedy algorithm (Algorithm 10) provides an allocation* $T_1, T_2, \ldots, T_m$ *to agents such that*

$$\sum_{i=1}^{m} v_i(T_i) \geq \frac{1}{2} \sum_{i=1}^{m} v_i(T_i^*).$$

The above theorem which follows from the result in [56] shows that the greedy algorithm is guaranteed to return an allocation that has an expected utility within 50% of the optimal value. We now provide an example where the expected utility of the greedy allocation is 61.87% of the optimal value to illustrate that this bound in not very loose.

Example 6.1.9 *Consider an instance consisting of two agents* A_1*,* A_2 *and two tasks* t_1*,* t_2*. Let the rewards of tasks* t_1, t_2 *be 17675 and 12490 respectively, and let the distance of each task from the base be 49. Let the probability of survival per unit distance for* A_1 *be 0.875, and that for* A_2 *be 0.88455. Let the agent values of* A_1 *and* A_2 *be 0.0036 and 0.075 respectively. The greedy allocation first allocates task* t_1 *to* A_1 *(in a greedy manner) and then has to allocate task* t_2 *to* A_2*. This results in an overall expected utility of 0.0331077. The optimal allocation for this instance is to allocate task* t_1 *to* A_2 *and to allocate* t_2 *to* A_1*. This allocation has an expected utility of 0.0535106. It can be seen that in this example, the ratio of utilities of the greedy allocation to the optimal allocation is 0.6187.*

We will now show that the sequence of allocations made by Algorithm 10 has an intuitive structure when we restrict attention to homogeneous agents (i.e., with $\psi_1 = \psi_2 = \cdots = \psi_m$ and $\theta_1 = \theta_2 = \cdots = \theta_m$). Note that in this case, the reward-to-risk ratio for each task is the same across all agents. For the homogeneous scenario, we present Algorithm 11.

Algorithm 11 Greedy Algorithm for the MARATA Problem in High-Risk Scenarios with Homogeneous Agents

procedure SGA-M(HOMOGENEOUS)(A, T, G)
  Initialize $T_1 = T_2 = \cdots = T_m = \emptyset$.
  Let $t_1, t_2, \ldots, t_n$ be the set of tasks, sorted in non-increasing order of their reward-to-risk ratios. For $j \in \{1, 2, \ldots, n\}$, let d_j denote the round-trip distance of task t_j from the base.
  for j from 1 to n **do**
    Let $i \in \{1, 2, \ldots, m\}$ be the index of the agent A_i for which the sum of round-trip distances in T_i is smallest (breaking ties arbitrarily).
    Set $T_i \leftarrow T_i \cup \{t_j\}$.
  end for
end procedure

Proposition 6.1.2 *Suppose that the probability of survival per unit distance ψ_i and agent value θ_i are homogeneous across all agents (i.e., $\psi_1 = \psi_2 = \cdots = \psi_m$ and $\theta_1 = \theta_2 = \cdots = \theta_m$). Furthermore, suppose that ψ_i and θ_i satisfy the condition* (6.4). *Let $\mathcal{A}^* = \{T_1^*, T_2^*, \ldots, T_m^*, T_\emptyset^*\}$ be an optimal allocation to the agents for the MARATA problem* (6.3) *under these assumptions. Then Algorithm 11 provides an allocation $T_1, T_2, \ldots, T_m$ to agents such that*

$$\sum_{i=1}^{m} v_i(T_i) \geq \frac{1}{2} \sum_{i=1}^{m} v_i(T_i^*).$$

Proof Since this proposition considers a special case of the scenario considered in Theorem 6.1.8 (pertaining to homogeneous agents), we know that Algorithm 10 provides an allocation that satisfies the condition in the proposition. We will show that Algorithm 11 is a special case of Algorithm 10 in the case of homogeneous agents. Note that the ordering of tasks in Algorithm 10 is arbitrary, and thus we can consider the tasks in non-increasing order of their reward-to-risk ratios. Since the agents are homogeneous, we also note that the MMG of each task (defined in Def. 6.1.1) is independent of the index of the agent that the task is assigned to. More specifically,

if we denote $\psi_i = \psi$ and $\theta_i = \theta$ for all $i \in \{1, 2, \ldots, m\}$, we denote the MMG of each task t_j by

$$\mu_j = r_j \psi^{d_j} - \theta(1 - \psi^{d_j}).$$

Next, consider the quantity $v_i(T_i \cup \{t_j\})$ for any agent A_i and task t_j. Since the tasks are being allocated in non-increasing order of their reward-to-risk ratio, and since we are considering a high-risk scenario where all tasks will be executed as one-shot cycles, the optimal location to add t_j to the walk currently executed by agent A_i will be *after* the one-shot cycles corresponding to the tasks in T_i (by Prop. 5.2.2). More specifically, suppose $W \in \mathcal{W}(T_i)$ is the optimal walk through the set of tasks T_i, with $\mathcal{C}(W) = \{W_1, W_2, \ldots, W_p\}$. Then, the optimal walk through the set of tasks $T_i \cup \{t_j\}$ will be given by $\bar{W}$, with $\mathcal{C}(\bar{W}) = \{W_1, W_2, \ldots, W_p, W_{t_j}\}$, where W_{t_j} is the one-shot cycle containing task t_j. By Lemma 6.1.5, the utility of the optimal walk $\bar{W}$ is given by

$$v_i(T_i \cup \{t_j\}) = E_i(\mathcal{C}(\bar{W})) = \left(\sum_{l=1}^{p} \psi^{\sum_{k=1}^{l-1} d_k} \mu_l \right) + \psi^{\sum_{k=1}^{p} d_k} \mu_j.$$

Thus, we have

$$\begin{aligned} v_i(T_i \cup \{t_j\}) - v_i(T_i) &= E_i(\mathcal{C}(\bar{W})) - E_i(\mathcal{C}(W)) \\ &= \psi^{\sum_{k=1}^{p} d_k} \mu_j. \end{aligned}$$

Thus, we see that the marginal gain in adding task t_j to T_i decreases with the sum of the round-trip distances of the tasks in T_i. Thus, when considering line 4 of Algorithm 10, the agent that has the smallest sum of round-trip distances in its current set of allocated tasks will yield the largest marginal gain when given task t_j. This yields line 4 in Algorithm 11. Thus Algorithm 11 is a special case of Algorithm 10, and the result follows. ■

6.2 Simulations

To understand how the greedy allocation compares to the optimal allocation in general, we carried out some simulations. The simulations were carried out on a

computer with an Intel (R) Core(TM) i5-8500 (3 GHz) CPU and 16 GB RAM. Rewards for tasks were generated randomly in the range 90000 to 100000 (from a discrete uniform distribution), and distances were generated randomly from the range 1 to 1000. Agent values and survival probabilities were selected so that all tasks will be executed as one-shot cycles and all tasks are high-rewards tasks. Figure 6.2 shows the plots comparing the expected utilities and runtimes of both the allocations when there are two agents. The x-axis shows the number of tasks allocated and the y-axis for the first plot shows the ratio of the expected utilities of the greedy and optimal allocation. The ratio of expected utilities was chosen as the metric as the range of values of expected utilities varied with problem instances. The y-axis for the second plot shows the average execution times to compute both the greedy allocation and the optimal allocation. For each case, the values were averaged over a 100 runs. Figures 6.3 and 6.4 shows a similar comparison for three and four agents respectively. We can see through the simulations that the greedy algorithm (in general) returns allocations that have expected utilities close to the optimal values. The execution time for the greedy algorithm is very small for large instances of the problems, whereas the execution time for finding the optimal allocation increases rapidly as the size of the problem instance increases.

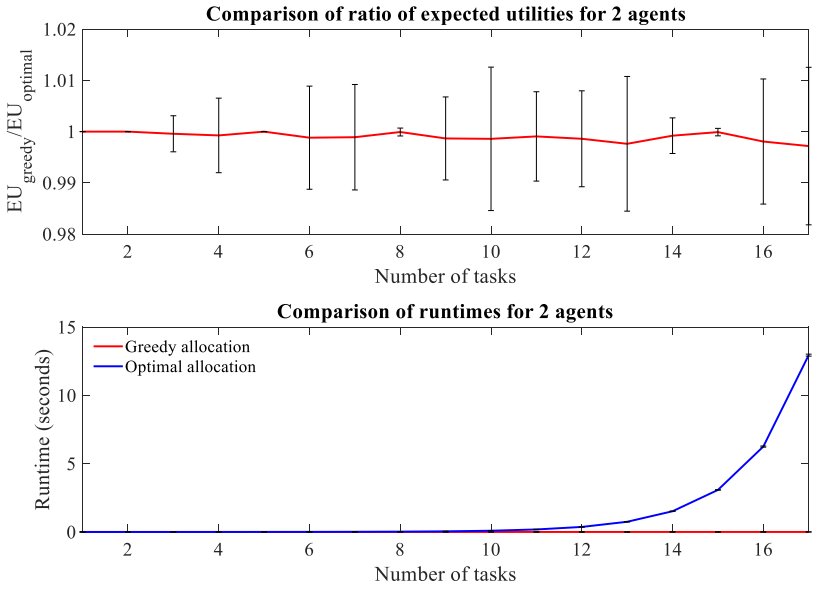


Fig. 6.2. Comparison of performance of greedy allocation and the optimal allocation when the number of agents is two.

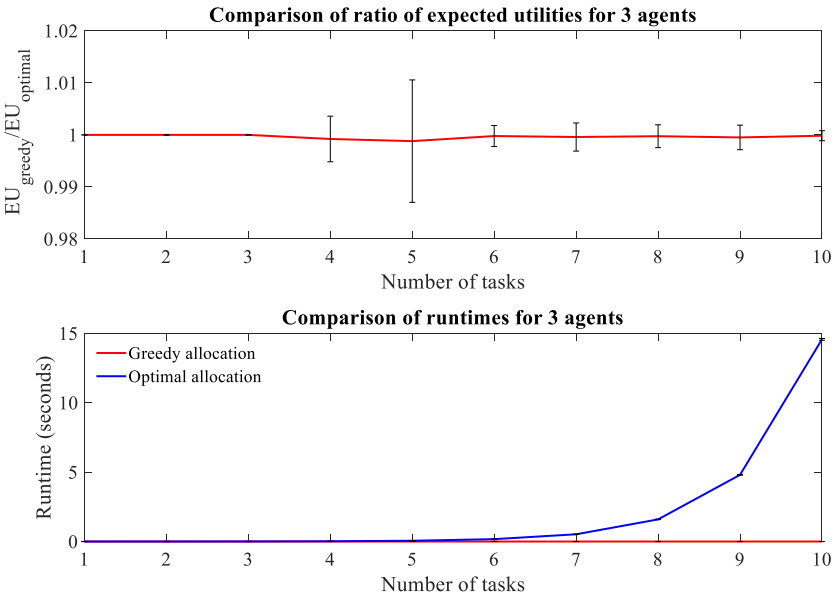


Fig. 6.3. Comparison of performance of greedy allocation and the optimal allocation when the number of agents is three.

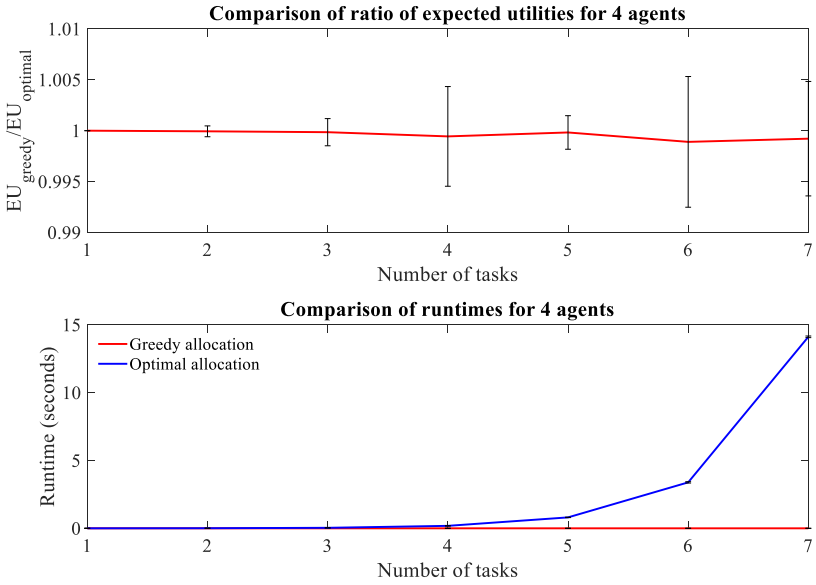


Fig. 6.4. Comparison of performance of greedy allocation and the optimal allocation when the number of agents is four.

6.3 Chapter Summary

In this chapter, we considered the risk-aware scenario consisting of multiple agents to collect the sensor measurement data, and defined the Multi-Agent Risk-Aware Task Allocation (MARATA) problem. We showed that the optimization problem is submodular in high-risk scenarios, and provided a greedy algorithm that returns an allocation within 50% of the optimal expected value. We also showed some simulations for the greedy allocation to understand how it compares against the optimal allocation.

7. SUMMARY AND FUTURE WORK

In this thesis, we looked at task allocation and routing of multiple heterogeneous agents. We looked at problems where functionally heterogeneous agents are required to complete tasks cooperatively while minimizing the maximum cost incurred by any agent. We considered the Heterogeneous Agent Cycle Problem (HACP), where we found tours for such agents located at a common start location. We provided a 5-approximation algorithm for HACP. We then considered a more general version of HACP, which we call the Heterogeneous Agent Path Problem (HAPP). In this problem, we relax the constraint that agents must have a common start location. In this framework, agents may have different start locations; furthermore, since the start locations are arbitrary, the agents are not constrained to rendezvous back to their start locations. This allows resilient and adaptive task allocation to these agents. We studied this problem and provided a 15-approximation algorithm.

In the second half of this thesis, we considered agent routing under "risk". That is, an agent faces a non-zero risk of destruction when it travels. We studied risk based routing for a single agent by studying the Single Agent Risk Aware Task Execution (SARATE) Problem. We characterized several key properties of the optimal policy under such a scenario. We also provided the optimal policy under the special case where the risk of destruction is very high. We then extended our analysis to scenarios with multiple agents. We show that the scoring scheme is submodular under sufficiently high risk conditions and show that the greedy algorithm is guaranteed to provide an allocation that has expected utility within 50% of the optimal value.

7.1 Future Directions

The work in this thesis explores some key areas on path planning for multi-agent systems. The following are some interesting areas for future research.

- Approximation algorithm for SARATE: The goal will be to provide algorithms with worst case performance bounds for the SARATE Problem that was considered in Chapter 5. This is challenging as the algorithm will have to take into consideration several factors such as the reward of a task, its associated location, risk, the value of the agent, etc, in order to find a solution. Furthermore, depending on these factors, some tasks may not be present in the optimal tour, which further adds to the complexity of the problem.

- Approximation algorithm for MARATA under lower risk scenarios: Our analysis provided a solution guaranteed to be within 50% of the optimal expected value under high risk scenarios. It would be interesting and challenging to study the system when the risk is not high enough for the scoring scheme to be submodular.

- Consider broader classes of policies for MARATA: In our analysis, we considered allocations under partition policies. In the future, it would be interesting to consider broader classes of policies where tasks may be allocated to multiple agents for redundancy in case of agent failures or where agents are allowed to wait to know the outcome of another agent's tour before starting it's own.

- Policies for SARATE and MARATA to handle a schedule of tasks: Consider a scenario where we are given a time-extended schedule of how tasks are going to appear. Our current analysis does not consider a task that is not yet available, but will appear by the time the agent travels to that task location. It would be interesting to extend the study to cater to such scenarios.

REFERENCES

REFERENCES

[1] T.-C. Au, B. Banerjee, P. Dasgupta, and P. Stone, "Multirobot systems," *IEEE Intelligent Systems*, no. 6, pp. 3–5, 2017.

[2] P. E. Rybski, N. P. Papanikolopoulos, S. A. Stoeter, D. G. Krantz, K. B. Yesin, M. Gini, R. Voyles, D. F. Hougen, B. Nelson, and M. D. Erickson, "Enlisting rangers and scouts for reconnaissance and surveillance," *IEEE Robotics Automation Magazine*, vol. 7, no. 4, pp. 14–24, Dec 2000.

[3] S. J. Trujillo, V. L. Ruiz, N. Esparza, J. A. Riley, K. C. Chu, and W. Bokhari, "Solar tracking system employing multiple mobile robots," Nov. 15 2016, US Patent 9,494,341.

[4] L. Bertuccelli, H.-L. Choi, P. Cho, and J. How, "Real-time multi-UAV task assignment in dynamic and uncertain environments," in *AIAA Guidance, Navigation, and Control Conference*, 2009, p. 5776.

[5] A. Sadeghi and S. L. Smith, "Heterogeneous task allocation and sequencing via decentralized large neighborhood search," *Unmanned Systems*, vol. 5, no. 02, pp. 79–95, 2017.

[6] J. Bae and W. Chung, "Heuristics for two depot heterogeneous unmanned vehicle path planning to minimize maximum travel cost," *Sensors*, vol. 19, no. 11, p. 2461, 2019.

[7] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[8] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," DTIC Document, Tech. Rep., 1976.

[9] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[10] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[11] T. Bektas, "The multiple traveling salesman problem: An overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209 – 219, 2006.

[12] M. R. Garey and D. S. Johnson, *Computers and intractability*. WH Freeman New York, 2002, vol. 29.

[13] E. M. Arkin, R. Hassin, and A. Levin, "Approximations for minimum and min-max vehicle routing problems," *Journal of Algorithms*, vol. 59, no. 1, pp. 1–18, 2006.

[14] D. Manerba, R. Mansini, and J. Riera-Ledesma, "The traveling purchaser problem and its variants," *European Journal of Operational Research*, vol. 259, no. 1, pp. 1–18, 2017.

[15] P. Toth and D. Vigo, *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[16] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[17] B. L. Golden, L. Levy, and R. Vohra, "The orienteering problem," *Naval Research Logistics (NRL)*, vol. 34, no. 3, pp. 307–318, 1987.

[18] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, no. 1, pp. 1–10, Feb. 2011.

[19] K. Sundar and S. Rathinam, "An exact algorithm for a heterogeneous, multiple depot, multiple traveling salesman problem," in *International Conference on Unmanned Aircraft Systems*. IEEE, 2015, pp. 366–371.

[20] 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.

[21] F. Bullo, E. Frazzoli, M. Pavone, K. Savla, and S. L. Smith, "Dynamic vehicle routing for robotic systems," *Proceedings of the IEEE*, vol. 99, no. 9, pp. 1482–1504, 2011.

[22] A. Prorok, M. A. Hsieh, and V. Kumar, "Adaptive distribution of a swarm of heterogeneous robots," *Acta Polytechnica*, vol. 56, no. 1, pp. 67–75, 2016.

[23] E. G. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. Stentz, "Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks," in *Proceedings International Conference on Robotics and Automation, ICRA*. IEEE, 2006, pp. 570–575.

[24] R. Kilgore, K. Harper, C. Nehme, and M. Cummings, "Mission planning and monitoring for heterogeneous unmanned vehicle teams: A human-centered perspective," in *AIAA Infotech@ Aerospace 2007 Conference and Exhibit*, 2007, p. 2788.

[25] Office of Naval Research, "Autonomous systems innovation summit final report," Office of Naval Research, Tech. Rep., 2008.

[26] Z. Xu and B. Rodrigues, "A 3/2-approximation algorithm for the multiple TSP with a fixed number of depots," *INFORMS Journal on Computing*, vol. 27, no. 4, pp. 636–645, 2015.

[27] W. Malik, S. Rathinam, and S. Darbha, "An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem," *Operations Research Letters*, vol. 35, no. 6, pp. 747–753, 2007.

[28] S. Yadlapalli, S. Rathinam, and S. Darbha, "3-approximation algorithm for a two depot, heterogeneous traveling salesman problem," *Optimization Letters*, vol. 6, no. 1, pp. 141–152, 2012.

[29] J. Bae and S. Rathinam, "A primal-dual approximation algorithm for a two depot heterogeneous traveling salesman problem," *Optimization Letters*, vol. 10, no. 6, pp. 1269–1285, 2016.

[30] G. N. Frederickson, M. S. Hecht, and C. E. Kim, "Approximation algorithms for some routing problems," in *Foundations of Computer Science, 17th Annual Symposium on.* IEEE, 1976, pp. 216–227.

[31] W. Yu and Z. Liu, "Improved approximation algorithms for min-max and minimum vehicle routing problems," in *International Computing and Combinatorics Conference.* Springer, 2015, pp. 147–158.

[32] G. Even, N. Garg, J. Könemann, R. Ravi, and A. Sinha, "Min–max tree covers of graphs," *Operations Research Letters*, vol. 32, no. 4, pp. 309–315, 2004.

[33] M. R. Khani and M. R. Salavatipour, "Improved approximation algorithms for the min-max tree cover and bounded tree cover problems," *Algorithmica*, vol. 69, no. 2, pp. 443–460, 2014.

[34] K. Zhang, E. G. Collins Jr, and D. Shi, "Centralized and distributed task allocation in multi-robot teams via a stochastic clustering auction," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 7, no. 2, p. 21, 2012.

[35] L. Wang, Z. Yu, Q. Han, B. Guo, and H. Xiong, "Multi-objective optimization based allocation of heterogeneous spatial crowdsourcing tasks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 7, pp. 1637–1650, 2018.

[36] M. Yu, V. Nagarajan, and S. Shen, "An approximation algorithm for vehicle routing with compatibility constraints," *Operations Research Letters*, vol. 46, no. 6, pp. 579 – 584, 2018.

[37] S. Rathinam, R. Ravi, J. Bae, and K. Sundar, "Primal-Dual 2-Approximation Algorithm for the Monotonic Multiple Depot Heterogeneous Traveling Salesman Problem," in *17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), S. Albers, Ed., vol. 162. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, pp. 33:1–33:13.

[38] J. Carlsson, D. Ge, A. Subramaniam, A. Wu, and Y. Ye, "Solving min-max multi-depot vehicle routing problem," *Lectures on global optimization*, vol. 55, pp. 31–46, 2009.

[39] M. Franceschelli, D. Rosa, C. Seatzu, and F. Bullo, "Gossip algorithms for heterogeneous multi-vehicle routing problems," *Nonlinear Analysis: Hybrid Systems*, vol. 10, pp. 156 – 174, 2013, special Issue related to IFAC Conference on Analysis and Design of Hybrid Systems (ADHS 12).

[40] A. Prasad, H.-L. Choi, and S. Sundaram, "Min-max tours for task allocation to heterogeneous agents," in *IEEE Conference on Decision and Control (CDC)*, Dec 2018, pp. 1706–1711.

[41] A. Prasad, H. Choi, and S. Sundaram, "Min-max tours and paths for task allocation to heterogeneous agents," *IEEE Transactions on Control of Network Systems*, 2020.

[42] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, 1992.

[43] C. H. Papadimitriou, "The Euclidean travelling salesman problem is NP-complete," *Theoretical Computer Science*, vol. 4, no. 3, pp. 237–244, 1977.

[44] W. Yu and Z. Liu, "Better inapproximability bounds and approximation algorithms for min-max tree/cycle/path cover problems," in *International Computing and Combinatorics Conference*. Springer, 2017, pp. 542–554.

[45] M. Turpin, N. Michael, and V. Kumar, "An approximation algorithm for time optimal multi-robot routing," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 627–640.

[46] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[47] J. K. Lenstra, D. B. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Mathematical programming*, vol. 46, no. 1-3, pp. 259–271, 1990.

[48] D. Feillet, P. Dejax, and M. Gendreau, "Traveling Salesman Problems with Profits," *Transportation Science*, vol. 39, no. 2, pp. 188–205, May 2005.

[49] N. Hazon, Y. Aumann, S. Kraus, and D. Sarne, "Physical search problems with probabilistic knowledge," *Artificial Intelligence*, vol. 196, pp. 26–52, Mar. 2013.

[50] C.-S. Liao and Y. Huang, "Generalized Canadian traveller problems," *Journal of Combinatorial Optimization*, vol. 29, no. 4, pp. 701–712, 2015.

[51] J. Hudack and J. C. Oh, "Multi-agent sensor data collection with attrition risk," in *26th International Conference on Automated Planning and Scheduling*, 2016.

[52] J. Hudack, "Risk-aware planning for sensor data collection," Ph.D. dissertation, Syracuse University, 2016.

[53] A. Prasad, J. Hudack, and S. Sundaram, "On optimal policies for risk-aware sensor data collection by a mobile agent," *IFAC-PapersOnLine*, vol. 52, no. 20, pp. 321 – 326, 2019, 8th IFAC Workshop on Distributed Estimation and Control in Networked Systems NECSYS 2019.

[54] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, ser. The MIT Press. MIT Press, 2009.

[55] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, "An analysis of approximations for maximizing submodular set functionsii," in *Polyhedral combinatorics*. Springer, 1978, pp. 73–87.

[56] B. Lehmann, D. Lehmann, and N. Nisan, "Combinatorial auctions with decreasing marginal utilities," *Games and Economic Behavior*, vol. 55, no. 2, pp. 270–296, 2006.

[57] J. Vondrák, "Optimal approximation for the submodular welfare problem in the value oracle model," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 2008, pp. 67–74.

[58] S. Dobzinski and M. Schapira, "An improved approximation algorithm for combinatorial auctions with submodular bidders," in *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, 2006, pp. 1064–1073.

[59] P. Segui-Gasco, H.-S. Shin, A. Tsourdos, and V. Segui, "Decentralised submodular multi-robot task allocation," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 2829–2834.

VITA

VITA

Amritha Prasad received her Bachelor's Degree in Avionics from the Indian Institute of Space Science and Technology (IIST) in 2011. She worked as a Scientist/Engineer at the Vikram Sarabhai Space Centre (VSSC) for the Indian Space Research Organization (ISRO) from 2011 to 2016 before joining the PhD program at Purdue University in 2016. At Purdue, she is a recipient of the Chintakindi L. (C. L.) and Sita C. Amba-Rao Fellowship, and the Estus H. and Vashti L. Magoon Award for Excellence in Teaching. She has also received the National Talent Search Scholarship awarded by the Govt. of India. Her research interests include multi-agent systems, routing problems, risk-aware systems, network science and algorithm design.